Advances in Lidar Point Cloud Segmentation for Automotive Applications

Segmenting Outside the Box

von der Fakultät für Elektrotechnik, Informationstechnik und Medientechnik
der Bergischen Universität Wuppertal genehmigte

Dissertation

---

zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften

von

Frederik Lenard Hasecke

aus

Solingen

Wuppertal 2023

Tag der Prüfung	08.11.2023
Hauptreferent	Prof. Dr.-Ing. Anton Kummert
Korreferent	Prof. Dr.-Ing. Bin Yang

Frederik Lenard Hasecke

Advances in Lidar Point Cloud Segmentation for Automotive Applications

Segmenting Outside the Box

Dissertation

Abstract

This dissertation presents a comprehensive study on novel lidar segmentation techniques and their applications.

The first part focuses on the development of neural networks and heuristic algorithms to perform tasks such as instance, semantic, and panoptic segmentation of lidar point clouds. These methods have made significant advancements and hold promise for various practical applications.

The second part introduces novel techniques in lidar augmentation and domain adaptation. These methods enhance the performance of segmentation algorithms, ensuring their robustness in the face of changes in sensor configuration and environmental conditions. Furthermore, they enable the training and application of segmentation networks with high effectiveness, even when dealing with limited or no annotations.

In the third part novel applications based on lidar segmentation are introduced. The applications include the creation of maps that represent the environment's geometric and semantic information, real-time object recognition on a single CPU core, and re-simulation environments designed specifically for training advanced driver assistance systems.

Furthermore, the techniques devised in the second part were adapted and applied in a public object recognition competition, where the proposed method secured the first-place position, further validating their effectiveness and superiority.

The research findings from this study serve as compelling evidence for the effectiveness and efficiency of the proposed methods. These methods hold significant practical value in real-world scenarios and contribute to the advancement of lidar segmentation for autonomous systems. By showcasing their applicability and impact, this work paves the way for further developments and improvements in the field, ultimately driving progress in autonomous systems.

Zusammenfassung

Diese Dissertation bietet eine umfassende Studie über neuartige Lidar-Segmentierungsverfahren und ihre verschiedenen Anwendungen.

Der erste Teil der Studie konzentriert sich auf die Entwicklung neuronaler Netze und heuristischer Algorithmen für verschiedene Lidar-Punktwolken-Segmentierungsaufgaben, einschließlich Instanz-, semantischer und panoptischer Segmentierung. Diese Methoden haben signifikante Fortschritte erzielt und weisen großes Potenzial für praktische Anwendungen auf.

Im zweiten Teil der Dissertation werden innovative Techniken zur Lidar Datenaugmentierung und Domänenanpassung vorgestellt. Diese Techniken verbessern die Leistung der Segmentierungsalgorithmen und gewährleisten ihre Widerstandsfähigkeit gegenüber Änderungen der Sensorkonfiguration und der Umweltbedingungen. Darüber hinaus ermöglichen sie ein effektives Training und den Einsatz von Segmentierungsnetzwerken, selbst bei begrenzten oder fehlenden Annotationen.

Im dritten Teil werden neue Anwendungen auf der Grundlage der Lidar-Segmentierung vorgestellt. Die Anwendungen umfassen die Erstellung von Karten, die die geometrischen und semantischen Informationen der Umgebung darstellen, Objekterkennung in Echtzeit auf einem einzigen CPU-Kern und Re-Simulationsumgebungen, die speziell für das Training fortschrittlicher Fahrerassistenzsysteme entwickelt wurden.

Weiterhin wurden die im zweiten Teil vorgestellten Techniken angepasst und erfolgreich in einem öffentlichen Wettbewerb zur Objekterkennung angewandt, wo sie den ersten Platz belegten, was ihre Effektivität und Überlegenheit weiter bestätigt.

Die in dieser Studie vorgestellten Forschungsergebnisse liefern überzeugende Beweise für die Wirksamkeit und Effizienz der vorgeschlagenen Methoden. Diese Methoden besitzen einen praktischen Wert in realen Szenarien und tragen zur Weiterentwicklung der Lidar-Segmentierung für autonome Systeme bei. Durch die Demonstration ihrer Anwendbarkeit und Wirkung schafft diese Arbeit die Voraussetzungen für weitere Fortschritte und Verbesserungen auf diesem Gebiet und treibt letztlich den Fortschritt bei autonomen Systemen voran.

Acknowledgement

I would like to express my deepest gratitude to my supervisor Prof. Dr.-Ing. Anton Kummert for his unwavering support, guidance, and encouragement even during the times when our papers were repeatedly rejected, his belief in my abilities kept me motivated and determined to persevere.

Furthermore I would like to express my gratitude to Dennis Müller and Christian Nunn for their support in helping me secure the scholarship for my PhD. I am deeply appreciative of the trust they placed in me and their belief in my abilities.

I am deeply grateful to Andre Paus and Lukas Hahn for giving me the chance to participate in the "@CITY" project that led to my masters thesis and my PhD. The trust they placed in me was crucial in starting my academic journey. I am forever appreciative for the opportunity to work with them.

I would like to extend my sincere gratitude to Lutz Roese-Koerner and Urs Zimmermann, the two Product Owners I had the pleasure of working with. I am deeply grateful for their guidance and support, and even more so the way they shielded me from the industry-related challenges as much as possible for me to focus on my research. Their mentorship and dedication have been invaluable and I am truly grateful for the opportunity to work with them. Their support has not only helped me to complete my research but also prepared me for the next step in my career.

I would also like to express my sincere gratitude to my colleagues and fellow PhD students, in alphabetical order, Ido Freeman, Lukas Hahn, Martin Alsfasser and Pascal Colling for the stimulating discussions, constructive feedback, and collaborative spirit throughout the duration of my PhD. Their support and camaraderie have been an integral part of my academic journey. I have learned a lot from all of them and their contributions have been invaluable. Their friendship and support have been a ever present motivation and inspiration throughout my PhD.

On a personal note, I would love to express my heartfelt appreciation to my parents Bogusia Hasecke and Jan Ulrich Hasecke, and my brother Jan Filip Tristan Hasecke for their unwavering encouragement throughout my journey. Their love has been a constant source of strength and inspiration.

And lastly, Rebecca, without your support in the past years I would not have been able to write these words. From the gentle support, to the harsh kicks in the butt you always knew what to do and what to say to bring me back on track. Your constant encouragement and belief in me have been my greatest source of motivation and strength. You are my confidante, and my best friend. Your support has been invaluable and I cannot express my gratitude enough. Your willingness to manage and take care of our important life matters, to give me the time and strength to focus on my research is truly admirable and I am forever grateful for your sacrifices. Your love and support have meant everything to me and I am truly blessed to have you in my life. More than anyone else, I am most thankful for you and the role you have played in my academic but more so my personal life.

Contents

1 Introduction

Autonomous vehicles, or self-driving cars, have the potential to revolutionize transportation and improve safety by eliminating human error. These vehicles use various sensors, such as cameras, radar, ultrasonic, and lidar, to constantly scan their surroundings and make decisions based on the collected data. By reducing the number of accidents caused by human factors such as distracted or impaired driving, self-driving cars can enhance road safety. Furthermore, they can provide greater mobility for elderly or disabled individuals who currently face transportation challenges [65]. However, to gain acceptance from society and regulators, self-driving cars must demonstrate a higher level of safety than human drivers, who already have a 99.999819% success rate in avoiding crashes [64]. To achieve this, self-driving cars must process vast amounts of data from sensors and other sources in real-time. This requires advanced machine learning algorithms and cutting-edge engineering to ensure safe operations in various scenarios.

Creating a precise and dependable model of the environment is one of the primary challenges in autonomous vehicle perception [146]. Humans create mental models of the environment using various cognitive and sensory mechanisms, a complex process that accurately represents the shape, position, properties, and context of all objects [170]. In contrast, lidar sensors, for example, generate a three-dimensional point cloud of the environment by emitting laser light and measuring the time it takes for the reflections to return [90]. Although the point cloud provides a rich and detailed representation of the environment, it lacks inherent meaning or semantics. Therefore, specialized algorithms and techniques are required to extract high-level abstract characteristics of the point cloud data that are relevant and meaningful to the task at hand.

Lidar segmentation plays a crucial role in enabling autonomous vehicles to "see" and navigate through their environment safely and efficiently, much like how human perception works. It involves breaking down a point cloud obtained from a lidar sensor into smaller parts that correspond to different objects or features within the environment. This is done by analyzing the characteristics of each point in the cloud, namely its location and intensity, to determine which object it belongs to. By segmenting the point cloud in this way, it becomes possible to detect and classify different objects within the environment, such as vehicles, pedestrians, and obstacles. Lidar segmentation adds the missing inherent meaning to the point clouds as explicit labels. These enable the autonomous vehicle's perception system to understand the layout and structure of the environment. This information is crucial for anticipating and responding to dynamic events, such as a pedestrian crossing the road.

To provide some context, lidar segmentation is a well-studied problem in the field of autonomous vehicles, and there are many approaches and algorithms that have been proposed in the literature to segment separate instances of objects in general [149, 45, 220, 99] and specific objects and parts of the environment [161, 162, 228, 7, 61, 132]. However, despite the progress made in this area, there are still several challenges that need to be addressed.

One of the main challenges is the complexity and variability of the real-world environment [146]. Lidar sensors can generate noisy and incomplete data due to factors such as occlusions, reflections, and interference from other sources [90]. Additionally, the geometry and appearance of objects can vary greatly depending on factors such as sensor type [30, 166, 42, 60], environment [121], weather [204], and seasonal changes [50]. These factors can make it difficult to develop robust and accurate algorithms that are able to handle a wide range of scenarios.

Another challenge is the need for real-time processing and decision-making [146]. Autonomous vehicles must be able to perceive and understand their environment in real-time to make safe and timely decisions. This requires lidar segmentation algorithms that are efficient and fast enough to process large amounts of data in real-time [45, 220, 99, 133].

Finally, there is a need for standardization and interoperability in algorithms that model the environment in autonomous vehicle perception. The integration of various sensor sources, such as lidar, camera, and radar data, is crucial for creating a more complete and accurate model of the environment. Combining different sensors using multi-modality and joint sensor reference frames allows for capturing complementary information, improving the overall perception system's reliability and safety [62]. However, the integration of different sensor sources is a complex task that requires advanced algorithms and techniques to handle the varied nature of data from different sensors. Therefore, developing accurate and robust algorithms that can effectively utilize information from different sensors is a critical challenge for the field of autonomous vehicle perception in general and especially for segmentation of different sensor data.

This thesis investigates the role of lidar segmentation in the perception system of autonomous vehicles and presents novel approaches to improve its accuracy and efficiency. The main research question driving this work is oriented towards the presented challenges:

How can lidar segmentation be utilized to enhance the capabilities and improve the safety of autonomous vehicles?

The ideas and methods proposed in this thesis are evaluated based on their practical application in the automotive industry. Lidar segmentation is a complex problem that involves balancing various conflicting goals, such as high performance and low cost or real-time run-ability in a vehicle and high fault tolerance. Furthermore, there is a trade-off between the benefits of three-dimensional dense information and its usability in navigation and decision algorithms. To address these challenges, this thesis focuses on three variations of the main research question:

- How can lidar segmentation be enhanced?
- How can the training cost of lidar segmentation be reduced?
- What are feasible use cases of lidar segmentation for autonomous vehicles?

1.1 Main Contributions

This thesis introduces novel approaches to enhance lidar segmentation performance, resulting in the extraction of valuable information from point clouds. The proposed algorithms and specialized techniques represent significant advances in the field, addressing critical challenges in autonomous vehicle perception systems. The research contributes to multiple aspects of autonomous vehicle perception, highlighting the potential of lidar segmentation to improve safety, reliability, and performance.

In Chapter 2, the fundamental concepts of lidar sensors, machine learning, and segmentation are outlined as they form the basis for the subsequent chapters and methods. Understanding these concepts is crucial to comprehending the methods presented in this thesis.

The main body of this thesis is divided into three parts, each addressing one of the formulated research questions.

In Part I, novel approaches for lidar segmentation are presented. These new methods were developed to address the question of how lidar segmentation can be enhanced in general. These methods include the *FLIC* (*Fast Lidar Image Clustering*) algorithm for real-time instance segmentation, the *RangePillars* network for semantic segmentation, and two new methods for panoptic segmentation.

In Part II, the focus shifts to reducing training costs for lidar segmentation and enhancing the robustness, thus aiming to answer the second formulated question "How can the training cost of lidar segmentation be reduced?". The *SAPCA* (*Structure Aware Point Cloud Augmentation*) method improves all metrics of semantic segmentation networks and outperforms state-of-the-art semi-supervised methods while drastically reducing the amount of data needed for a well performing segmentation network. Furthermore a novel domain adaptation method is presented, which improves the generalization and robustness of semantic segmentation enabling the successful reuse of already labeled data of a given sensor for entirely new lidar sensors.

The last part of this thesis, Part III, concludes by exploring new potential applications of lidar segmentation in autonomous vehicles, advanced driver assistance systems and the application of segmentation techniques for object detection tasks. These novel applications are proposals to answer the third question of feasible use cases of lidar segmentation for autonomous vehicles.

Overall, this thesis presents novel methods of lidar segmentation, techniques for lidar segmentation, and applications derived from lidar segmentation. The proposed methods are rigorously evaluated using real-world data in challenging environments, with a focus on improving performance and runtime, among other factors. This work emphasizes the importance and value of lidar segmentation not only in the context of autonomous vehicle perception, but also in other fields, and provides valuable insights and suggestions for future research and development in this area.

1.2 Publications

Parts of this dissertation have been published in the following peer-reviewed conference articles (main contributor):

Frederik Hasecke, Lukas Hahn, and Anton Kummert
"FLIC: Fast Lidar Image Clustering"
10th International Conference on Pattern Recognition Applications and Methods (ICPRAM), 2021
Best Student Paper Award
Covered in Chapter 3

Frederik Hasecke, Martin Alsfasser, Anton Kummert
"What Can be Seen is What You Get: Structure Aware Point Cloud Augmentation"
33rd IEEE Intelligent Vehicles Symposium (IV), 2022
Covered in Chapter 6

Frederik Hasecke, Pascal Colling, and Anton Kummert
"Fake it, Mix it, Segment it: Bridging the Domain Gap Between Lidar Sensors"
12th International Conference on Pattern Recognition Applications and Methods (ICPRAM), 2023
Covered in Chapter 7

Two proposed methods in this thesis were part of collaborations. Own contributions are listed in each case. The two approaches were published in the following conference article and patent application:

Lukas Hahn, Frederik Hasecke, Anton Kummert
"Fast Object Classification and Meaningful Data Representation of Segmented Lidar Instances"
23rd IEEE International Conference on Intelligent Transportation Systems (ITSC), 2020
Covered in Chapter 5.2 and Chapter 8.2

Own contributions:

- Major contributions to the idea of the content of the article.
- Implemented and conducted major parts of the experiments.

Lukas Hahn, Maximilian Schaefer, Kun Zhao, Frederik Lenard Hasecke, Yvonne Schnickmann, Andre Paus
"Detection System for Predicting Information on Pedestrian"
US Patent US20220242453A1, 2022
Partly covered in Chapter 8.2

Own contributions:

- Contributions to the idea of the content of the patent.
- Minor contributions to the literature research and review.

Furthermore, a part of this thesis has been published as a winning challenge submission report, which was not peer-reviewed prior to its publication on the workshop website. Additionally, there are two more patents, which are still pending at the time of writing.

Frederik Hasecke and Anton Kummert
"Report on the LiDAR Self-Supervised Learning Challenge: Learning From a Limited Amount of High-Resolution LiDAR Data"
ECCV workshop on 3D Perception for Autonomous Driving, A workshop at the European Conference on Computer Vision (ECCV), 2022
First Place Winner
Covered in Chapter 8.4

Frederik Lenard Hasecke, Sönke Behrends
"Method, Device, and Computer Program for Determining a Change in Position and/or Orientation of a Mobile Apparatus"
US Patent US20220217499A1, 2022
Covered briefly in Chapter 8.1

Own contributions:

- Major contributions to the idea of the patent.
- Implemented and conducted all experiments.
- General editing of the patent draft.

Frederik Lenard Hasecke, Moritz Luszek
"Data Structure for Efficient Training of Semantic Segmentation Models"
EU Patent 23154924.7, 2023
Covered in Chapter 8.1

Own contributions:

- Major contributions to the idea of the patent.
- Implemented and conducted all experiments.
- General editing of the patent draft.

Finally, it must be mentioned that parts of the work in this dissertation already had their roots in the authors master's thesis. Parts related to it can be found in CHAPTERS 3, 5.2 and 8.2.

> **Frederik Hasecke**
> "Extended Object Detection and Classification with Combined Lidar and Camera Sensor Data"
> *Master's Thesis, 2020*

2 Fundamentals

This chapter provides an overview of the fundamental concepts related to lidar sensors, machine learning, and segmentation. The goal is to provide a strong foundation for the subsequent chapters and methods presented in this thesis.

It is important to note that some of the formulations and variable namings introduced in this chapter may not be directly transferred across chapters. For example, the distance d represents the distance between the lidar sensor and an object in Chapter 2.1, but in Chapter 5, it refers to the distance between two lidar points, independent of the lidar sensor's position.

In this chapter, the basic principles of lidar sensors, including how they work, their components, and the different types of lidar sensors, will be introduced. The basics of machine learning and segmentation, including different approaches to machine learning, neural networks, and popular segmentation methods, will also be covered. Throughout this thesis, consistency and clarity are ensured by using a notation convention in which vectors are denoted in **bold** letters.

2.1 Lidar Sensor Fundamentals

The work presented in this thesis is based on automotive lidar sensors. This section will briefly present the underlying principles as well as the most common types of such sensors used in the automotive industry.

Lidar sensors are a type of remote sensing technology that uses lasers to measure the distance to objects. Lidar sensors are commonly used in autonomous vehicles for perception tasks and feature a transmitter module and a receiver module that allow them to function mostly independent of ambient conditions. They measure depth information using the time of flight principle:

$$d = \frac{c_0 \cdot t}{2}, \tag{2.1}$$

where the time $t \in \mathbb{R}_+$ is measured between the emission of a light pulse and the reception of its reflection after it reflected of a target. The flight time is multiplied by the speed of light c_0 in $\frac{m}{s}$, to calculate travel distance. This total distance is divided by two to receive the distance d in meters (m) to the object, as the light travels to the object and back again [90].

The returns from a laser are not represented as discrete values, but rather as continuous values because the response to short light pulses that are reflected by an object and received back by the sensor follows a Gaussian distribution, which is a continuous probability distribution characterized by a bell-shaped curve [90]. The mean μ and variance σ^2 of the distribution are dependent on various factors such as the distance of the object and atmospheric conditions. This is shown in Figure 2.1 *a*.

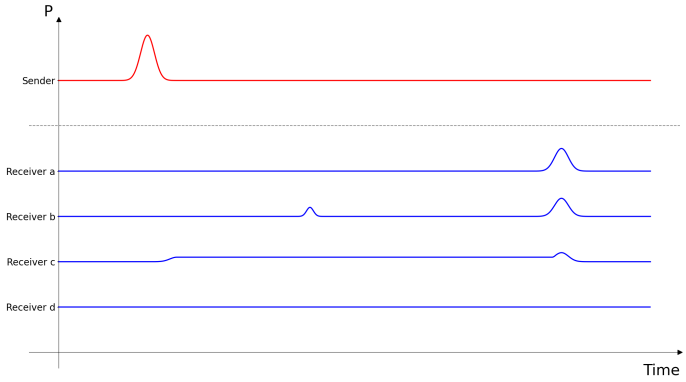

Figure 2.1.: Pulse Response of a Lidar at Different Conditions. The same impulse (*red*) is sent out at the same object with four different environmental conditions: A simple pulse response with a single target (*a*), multiple responses due to a partial reflection by a raindrop (*b*), a soft pulse response caused by a cloud of condensation (*c*) with a weaker target object response. And lastly a missing response due to the absorption and scattering of the light by particles in the air (*d*).

However, there can also be several peaks resulting from a single light pulse. This occurs when a laser beam is reflected back by multiple objects. This can happen, for example, through a raindrop or a pane of glass which reflects part of the laser beam back to the receiver, while most of the emitted beam hits the target behind the quasi-transparent target. Figure 2.1 *b* shows the Gaussian distribution of such a scenario. Likewise, each emitted laser beam has a cross-sectional area of the beam that depends on the manufacturer. When hitting a solid target with only a part of the cross section, only a part of the beam is reflected to the receiver. A second return is caused by the next object the rest of the beam hits. Some lidar sensor manufacturers therefore allow the output of multiple depth values per emitted laser beam to measure these multiple targets. Automotive lidar manufacturers usually provide two returns, the strongest peak as well as the last measurable peak [103, 199], airborne lidar even offer up to five return values [142]. The pulse response can also be a "soft" response or no response at all due to environmental influences such as increased back-scattering from particles in the air and absorbed light in the transmission respectively. These responses are shown in Figure 2.1 *c* and *d*.

The intensity of the return peaks from a laser can reveal important information about the reflectivity of target objects. This intensity is measured in watts (W), the unit of power or radiant flux, representing the rate at which energy is transferred. The incident power of the laser determines the rate at which it emits electromagnetic radiation, typically in the form of light. However, when the laser light encounters an object, only a portion of it is reflected back, resulting in a decrease in the reflected power compared to the emitted power. This is due to geometric influences, but also factors such as absorption and scattering. To account for these influences, the general lidar equation is used in atmospheric research [204]. In its simplest form, the equation is:

$$P_d = K\, G_d\, E_d, \tag{2.2}$$

where the received power P measured in W reflected from a distance d is composed by three factors: lidar system performance K, geometric property G_d at distance d, and the range-dependent environmental back-scattering and transmission coefficient E_d [204]. The same equation is used for automotive lidar sensors [194], as the equation takes the differences in range and environmental conditions into account.

The system constant K depends entirely on a priori values: The average power of a single laser pulse P_0 in W, the temporal pulse length τ in seconds (s), the area of the receiver module $A_{receiver}$ in square meters m^2, the overall unitless system efficiency η and the speed of light c_0 in $\frac{m}{s}$.

$$K = P_0 \frac{c_0 \tau}{2} A_{receiver}\, \eta. \tag{2.3}$$

The incident power and wavelength of automotive lidar sensors can vary depending on the specific type of sensor and the manufacturer. Both are known, predefined properties. Automotive lidar sensors typically use low-power lasers with wavelengths in the near-infrared range. The exact power of the laser beam can vary, but most automotive lidar sensors use relatively low power to minimize the potential risk to human eyesight [90]. The wavelength is usually 905 nanometers (nm) or 1,550 nm [129]. These wavelengths are outside the range of human vision and are considered safe for use in automotive applications. 1,550 nm lasers have better eye safety and longer detection range than 905 nm lasers, but they are approximately 145× more absorbed by the atmosphere, making them $4-5\times$ worse in detecting targets in rain and fog than their 905 nm counterparts. Moreover, 1,550 nm lasers have 97% less reflectivity in snow and consume on average more than ten times as much power compared to 905 nm lasers [137]. Therefore, the choice of laser wavelength in lidar systems involves trade-offs between eye safety, detection range, and targeted environments.

The geometric factor, G_d, present in Equation 2.2, incorporates both the surface area of the target, A_d (in square meters), at a given distance d, and the dimensionless reflectance factor, κ, of the object. This relationship is expressed as:

$$G_d = \frac{A_d}{d^2} \cdot \kappa. \tag{2.4}$$

Lidar operates by emitting a narrowly focused light beam towards a target, with the beam's dispersion reflected in the surface area of the target, A_d. When the beam strikes the target, it scatters in all directions, forming a spherical pattern of light. The lidar receiver captures only a fraction of this scattered light, and due to the spherical scattering, the captured light decreases as the square of the distance from the target, d.

To elaborate, the decay in signal strength with increasing distance adheres to a quadratic law, since the receiver intercepts only a portion of the light scattered from a sphere with radius d. This intercepted fraction is directly proportional to the solid angle, $\frac{A_d}{d^2}$, which symbolizes the lidar system's field of view for scattered light at a given distance d [90].

The geometric factor of the lidar distinctly varies from the doubled inverse-square law typically found in light sources or radar systems. These systems feature two-way propagation, causing spherical scattering during both the outbound and return journeys. This bi-directional dispersion considerably affects the received light or radar signal, culminating in a quartic dependency on distance in the radar equation, denoted as $1/d^4$.

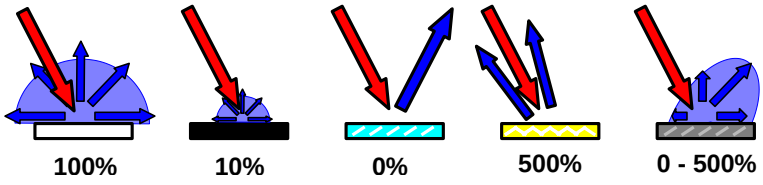

Figure 2.2.: **Reflectivity of Different Surfaces.** An ideal white Lambertian surface reflects 100% of the light that it is struck by in a diffuse manner, meaning that the light is reflected equally in all directions. In contrast, a dark Lambertian surface absorbs most of the light that it is struck by and reflects only a small portion of it diffusely and uniformly in all directions. A specular surface, on the other hand, reflects light in a specific direction, resulting in no emitted light from the lidar sensor reaching the receiver. Retro-reflectors are surfaces that are designed to reflect light back to its source through multiple reflections, and therefore tend to have higher reflectance compared to a Lambertian surface. In the real world, most surfaces have a combination of gray Lambertian and specular properties, with a varying degree of reflectance towards the receiver module.

The reflectance κ in Equation 2.2 is a measure of the fraction of incident light that is reflected by a surface. It is typically represented as a value between 0 and 1, where 0 indicates that the surface absorbs all incident light and 1 indicates that the surface reflects all incident light. Reflectance is similar to reflectivity, but it is usually defined for a specific angle of incidence and wavelength of light. For example, the reflectance of a surface at a specific angle of incidence θ in radians and wavelength λ in m is defined as:

$$\kappa(\theta, \lambda) = \frac{I_r(\theta, \lambda)}{I_i(\theta, \lambda)}, \tag{2.5}$$

where $\kappa(\theta, \lambda)$ is the reflectance of the surface at angle θ and wavelength λ, $I_r(\theta, \lambda)$ is the intensity of the reflected light at that angle and wavelength in watts per square meter ($\frac{W}{m^2}$), and $I_i(\theta, \lambda)$ is the intensity of the incident light at that angle and wavelength. The intensity of the reflected light can also be entirely different for the same inclination angle and the same wavelength given the influence of the surface property of the target object. In the case of a Lambertian reflection, i.e., an equally strong reflection in a semicircle independent of the angle of incidence, the reflectance for a defined inclination angle and wavelength is given by

$$\kappa = \frac{\Gamma}{\pi}, \tag{2.6}$$

where Γ is the reflection property of the target. Specular surfaces, like glass or polished metal, have a very low reflectance at all angles except for the angle at which light is reflected off them. This means that these types of surfaces have a high degree of reflectivity only when viewed from the angle at which light is reflected off. In Figure 2.2, the influence of various surfaces on the reflection intensity is shown. In reality there is no known material that exhibits a true Lambertian property [106].

The last term of the lidar Equation 2.2, the environmental term E_d, consists of the back-scattering β_d and the transmission α_d, both of which depend on the distance d, which furthermore corresponds to the length of the volume that the laser beam traverses. Further, both terms depend on the size, number, nature, refractive index, and shape of all particles within the volume that the laser beam traverses. In sunny weather with little dust and moisture in the air, these two terms have a neglectable influence on the final reflection intensity, while they dominate in rainy, foggy and polluted environments. This

is also shown in Figure 2.1 for three receivers b, c and d in which a laser beam traverses rain, a cloud of fog and a heavily polluted environment respectively. In summary, the lidar equation can be written as follows

$$P_d = P_0 \, \frac{c\tau}{2} A_{receiver} \, \eta \, \frac{A_d}{d^2} \kappa_{\theta,\Gamma} \, \beta_d \, \alpha_d. \tag{2.7}$$

If the system parameters that are known in advance are summarized as c_0

$$c_0 = P_0 \, \frac{c \, \tau \, A_{receiver} \eta}{2}, \tag{2.8}$$

and fixed variables such as the wavelength are omitted, the only variables that remain are the light fall-off, the target surface reflectance and the environmental influence

$$P_d = c_0 \underbrace{\frac{A_d}{d^2}}_{\text{fall-off}} \overbrace{\kappa_{\theta,\Gamma}}^{\text{reflectance}} \underbrace{\beta_d \, \alpha_d}_{\text{environment}} . \tag{2.9}$$

The reflection intensity values are usually normalized by the distance and quantized as an 8-bit value range. The resulting intensity values are therefore a combination of the inclination angle of the laser beam θ, the reflection property Γ of the target, and the prevailing environmental conditions β_d and α_d for the back-scattering and transmission of the traversed volume

$$\underbrace{\frac{P_d \, d^2}{c_0 \, A_d}}_{\text{quantized intensity}} = \overbrace{\kappa_{\theta,\Gamma}}^{\text{reflectance}} \underbrace{\beta_d \, \alpha_d}_{\text{environment}} . \tag{2.10}$$

Most surfaces in the real world exhibit a mixture of Lambertian and specular properties, due to which the relative intensity value can not draw direct conclusions onto the surface properties of the targets without the entrance angle and the surface orientation of the target.

The points described so far refer to the functionality of individual lidar modules. A full lidar sensor is a complete system that includes multiple lidar modules, as well as additional components such as a processor, memory, and a communication interface. The full lidar sensor uses the data from the individual lidar modules to create a three-dimensional map of the environment. This is usually done by moving parts within the lidar sensor, that move the lidar modules around a defined point of origin and measure the distance to a relative angle offset of a defined zero angle position in the vertical and horizontal position. In the following section the most common types of lidar sensors are described in more detail.

2.1.1 Automotive Lidar Sensors

There are two main types of lidar sensors used in the automotive industry: mechanical lidar and solid-state lidar [218, 118].

Mechanical lidar sensors usually use a spinning module to sweep multiple stacked laser beams across the surrounding environment. The emitter-receiver modules are typically mounted on a rotating

platform that spins around a central axis to capture continuous recordings of the entire environment in a 360° panoramic view in a sweeping pattern [118, 134, 218].

Solid-state lidars have multiple implementation methods but share the underlying principle of little to no moving parts. In general there are four solid-state lidar methods: Microelectromechanical systems, flash lidar, optical phase array and frequency-modulated continuous wave lidar [118].

- **Microelectromechanical lidar sensors (MEMS)** use tiny mirrors, that can be tilted by applying a voltage. This allows the system to scan the environment without using mechanical scanning hardware. The receiver aperture which determines the signal-to-noise ratio, is typically quite small for MEMS systems. To scan the environment in multiple dimensions, multiple mirrors are used.
- **Flash lidar** uses a single large laser pulse to illuminate the environment and a focal plane array of photo-detectors to capture the reflected light. This allows it to capture the entire scene in a single image which is faster than the mechanical scanning method used by other lidar sensor systems but requires a laser with a power output much higher than other lidar types and is blinded by retro-reflectors, rendering it useless in an automotive context.
- **Optical phased array (OPA)** lidar is similar to phased-array radar. It uses an optical phase modulator to control the phase and amplitude of light waves passing through the lens which allows the system to "steer" the laser beam without mechanical moving parts. This makes OPA lidar more reliable and efficient than other types of lidar.
- **Frequency-Modulated Continuous-Wave (FMCW)** lidar uses brief chirps of frequency-modulated laser light to measure both distance and velocity. This method is simpler in terms of computational load and optics compared to other types of lidar, but the chirp generation adds complexity.

The sensors of the datasets [87, 39, 13, @112] used in this thesis use mechanical lidar sensors [196, 197, 199, 103] and, most likely, microelectromechanical lidar sensors [@111]. The latter is not confirmed, as the sensor manual is not available at the time of writing.

In general, the type of lidar sensor used in a particular application will depend on factors such as the required range, field of view, and resolution, as well as the need for real-time processing and the constraints of the system. Rotating lidar sensors may be more suitable for applications that require fast scanning and a wide field of view, while solid state lidar sensors may be better for applications that require high resolution and a compact physical form.

2.2 Deep Learning

"Deep learning is a type of machine learning that involves using artificial neural networks to learn from data. These neural networks are composed of multiple layers of interconnected nodes which process and transform the input data into a more useful form. The goal of deep learning is to allow the model to automatically learn and improve from experience, without the need for explicit programming. This is achieved by training the model on large amounts of labeled data and using powerful computational resources to optimize the model's performance. Deep learning has been used

to achieve state-of-the-art performance in many fields, including computer vision, natural language processing, and speech recognition."

It may not be surprising to the reader that the previous paragraph, was actually written by an artificial neural network in response to the prompt "What is Deep Learning?" [176]. This is due to the fact that deep learning has a transformative effect on numerous fields, and has already fundamentally changed the way we approach many problems.

Deep learning also had a significant impact on lidar segmentation and other lidar perception tasks. Previously, hand-crafted features and heuristic algorithms were the primary methods used for tasks related to automotive and robotics perception. However, deep learning has now surpassed these approaches in nearly every category. The following section will provide a brief overview of deep learning, including an introduction to feedforward neural networks, convolutional neural networks, and deep learning neural networks specifically designed for segmentation. In the final section of this chapter, the use of lidar data with neural networks is briefly discussed, highlighting the integration of three-dimensional point clouds with deep learning approaches.

2.2.1 Feedforward Neural Networks

A feedforward neural network is a machine learning algorithm inspired by the human brain. It has multiple layers of interconnected neurons, where each neuron receives input from other neurons and produces an output that is passed on to the next layer. The output of the final layer is the output of the entire network. Supervised tasks of feedforward neural networks involve using input data to produce an associated desired target output. The neurons in the network are trained to give the desired output for a given input by adjusting their weights and biases in order to minimize the error between the predicted output and the desired output. This process is known as training the network. Once the network has been trained, it can use the learned relationships between input and output to predict the desired output for new input data.

In more detail, the inputs to each neuron of a feedforward neural network are combined with weights and biases to produce the output of each neuron. This is shown by the following equation:

$$\hat{y} = f(w_1 * x_1 + w_2 * x_2 + ... + w_n * x_n + b), \tag{2.11}$$

where $x_1, x_2, ..., x_n$ are the input values, $w_1, w_2, ..., w_n$ are the weights, b is the bias, f is the activation function and $\hat{y}$ is the predicted output of the neuron.

The activation function determines the output of each neuron in the network. It is typically a non-linear function that maps the input to the output in a way that allows the network to learn complex patterns in the data. A common activation function is the *sigmoid* function which is defined as

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{2.12}$$

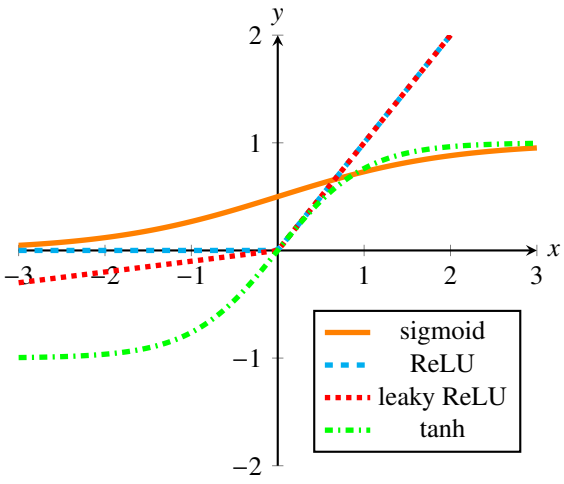

Figure 2.3.: Common Activation Functions of Neural Networks. The *sigmoid* activation function is plotted in orange, the *ReLU* in dashed cyan, the *leaky ReLU* in dotted red and the *tanh* in dash-dotted green.

where x is the input to the activation function and e is the base of the natural logarithm. There are several other commonly used activation functions in neural networks, such as the *hyperbolic tangent* (*tanh*) function

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{2.13}$$

the *rectified linear unit* (*ReLU*) function

$$f(x) = \max(0, x) \tag{2.14}$$

and the *leaky ReLU* function

$$f(x) = \max(a \cdot x, x) \tag{2.15}$$

where a is a small constant value, e.g. 0.1.

All four activation functions are plotted in Figure 2.3. Each of these activation functions has its own characteristics and advantages, and the choice of which activation function to use can depend on the specific problem and the type of data that is being processed. In general, the *sigmoid* function is often used in the output layer of a network when the output is a probability, while the *ReLU* function is often used in the hidden layers of a network to improve the speed and performance of the network [89].

During training, the weights and biases of a neural network are adjusted in order to reduce the error between the predicted output and the desired target. This is done by computing the gradients of the loss function with respect to the weights and biases using the backpropagation algorithm. An optimization algorithm such as *stochastic gradient descent* (*SGD*) is used to update the weights and biases based on these gradients. This process is repeated for multiple iterations until the network has been fully trained and the error no longer decreases.

The error of the predicted output $\hat{y}$ to the desired target y is evaluated by the loss function L which produces a scalar value that represents the magnitude of the error. The *mean squared error* (*MSE*) loss is a common loss function for regression tasks

$$L(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2. \tag{2.16}$$

The *binary cross-entropy* loss for binary classification tasks is

$$L(\hat{y}, y) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right] \tag{2.17}$$

and the *categorical cross-entropy* loss for multi-class classification tasks

$$L(\hat{y}, y) = -\sum_{c=1}^{C} y_c \log(\hat{y_c}), \tag{2.18}$$

where n is the number of samples, C is the number of classes, and y_c is the true label for class c.

Regardless of the task and chosen loss function, the gradient of the loss is computed with respect to the final output of the network

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial \hat{y}}. \tag{2.19}$$

This gradient is then propagated backwards through the network, layer by layer, using the chain rule of calculus to compute the gradients of the loss with respect to the weights and biases at each layer. For a given layer l with input x, output $\hat{y}$, weights w, and biases b, the gradient of the loss with respect to the weights can be computed as

$$\frac{\partial L}{\partial w_l} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_l}, \tag{2.20}$$

and the gradient of the loss with respect to the biases can be computed as

$$\frac{\partial L}{\partial b_l} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b_l}. \tag{2.21}$$

These gradients can then be used by an optimization algorithm such as *SGD* to update the weights and biases of the network and reduce the error. The weights and biases are updated according to the following equations:

$$w \leftarrow w - \eta \cdot \frac{\partial L}{\partial w} \tag{2.22}$$

$$b \leftarrow b - \eta \cdot \frac{\partial L}{\partial b} \tag{2.23}$$

where η is the learning rate which determines the size of the step taken in the direction of the gradient during each iteration of training. A larger learning rate can lead to faster convergence, but can also result in the algorithm overshooting the minimum of the loss function and failing to converge. A smaller learning rate results in a more stable and reliable convergence, but can fail to reach the optimal solution in a given number of iterations, or worse, settle in a local minimum. Finding an

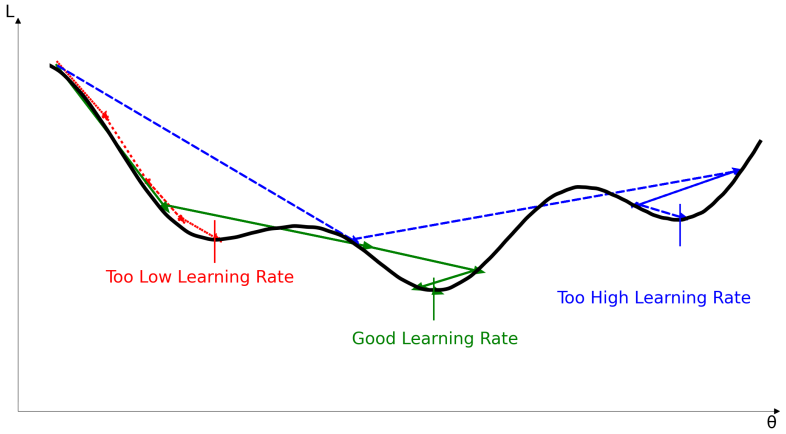

Figure 2.4.: Influence of Learning Rates on Finding an Optimal Solution. The loss surface plot shows the scalar value of the loss function L_θ as a function of a set of parameters summarized as θ. The results of three different optimization runs starting from the same point are shown. The dotted red line represents a run with a low learning rate which lands in an early local minimum. The green line represents a run with a medium learning rate which successfully reaches the global minimum. The dashed blue line represents a run with a high learning rate which overshoots the global minimum and lands in a high local minimum.

appropriate learning rate is an important part of training a neural network. FIGURE 2.4 shows the influence of learning rates to find the optimal solution.

One of the most popular gradient-based optimization algorithms is *stochastic gradient descent* (*SGD*). In *SGD*, the error is calculated for each training sample, and the weights and biases are adjusted based on the gradients for this error. This process is repeated for each training sample, and the weights and biases are updated on each iteration. This allows the algorithm to make rapid progress towards the optimal solution, but it also means that the algorithm can be sensitive to the order in which the data is presented.

To improve the performance of *SGD*, a number of variations and enhancements have been developed. One example is *mini-batch SGD* which uses a small batch of examples to calculate the error and update the weights and biases; *momentum SGD* which adds a momentum term to the update to help the algorithm escape from local minima; and *adaptive learning rate SGD* which adjusts the learning rate based on the current performance of the algorithm [89].

The training process is repeated for multiple epochs, where an epoch is a complete pass through the training dataset. After each epoch, the error is typically calculated to evaluate the performance of the network. The training process is stopped when the error reaches a satisfactory level or when a predetermined number of epochs has been reached.

The parameter space of a deep neural network is extremely high-dimensional, with potentially millions or even billions of parameters. The optimization of such a high-dimensional space is a challenging task, especially since the loss function is often non-convex, meaning it has many local minima that can trap the optimization algorithm [131].

Feedforward neural networks are versatile and can be used for a wide range of tasks, including regression [125], classification [184], and dimensionality reduction [67]. Unfortunately, they still

have some limitations, such as their inability to process data with complex temporal or spatial dependencies and their susceptibility to overfitting. Overfitting refers to the situation where the model has been trained too well on the training data, resulting in poor generalization to new data [89]. Overfitting can occur when a model is excessively complex, such as having too many parameters relative to the amount of training data. It can also occur when a model is trained for too long, leading to the model learning the noise in the training data instead of the underlying relationship. To prevent overfitting, it is important to use techniques such as regularization and early stopping.

Regularization is a technique that adds a penalty term to the loss function which helps to prevent the model from becoming too complex. Commonly used regularization techniques include *L1* and *L2 regularization*.

L1 regularization, also known as *Lasso regularization*, adds a penalty term to the loss function that is proportional to the absolute value of the weight coefficients. This has the effect of pushing some of the weight coefficients towards zero, effectively removing those features from the model [95]. *L2 regularization*, also known as *Ridge regularization*, adds a penalty term to the loss function that is proportional to the square of the weight coefficients [208]. This has the effect of "shrinking" the weight coefficients which can help to reduce overfitting. L1 regularization is defined as

$$L1(w) = \lambda \sum |w|, \tag{2.24}$$

while the definition of L2 regularization is

$$L2(w) = \lambda \sum w^2, \tag{2.25}$$

where w is a vector of the weight coefficients, and λ is a hyperparameter that controls the strength of the regularization. In both cases, the regularization term is added to the loss function which is then minimized during training.

Another option to reduce overfitting is called *Dropout* [185]. This is a technique that randomly drops some of the connections between neurons in the network during training. This can help to prevent the network from becoming too reliant on any one feature which can lead to overfitting [185].

Finally the technique *early stopping* [159] involves monitoring the performance of the model on a validation set during training, and stopping the training process when the performance on the validation set starts to degrade. This can help to prevent the model from overfitting to the training data.

Recent studies suggest that over-parameterized models in deep learning often optimize and generalize well, and prevent overfitting, despite not following the conventional learning curve [152]. The cause of this phenomenon remains unknown, but some studies suggest that over-parameterized classifier layers lead to overfitting while over-parameterized hidden layers prevent the same [231]. Others claim that stochastic gradient descent with random initialization introduces an implicit regularization effect, biasing the learning process towards solutions that generalize well to unseen data, thereby reducing overfitting in over parameterized neural networks [135].

Feedforward neural networks are versatile and can process various data formats, including numerical, text, and categorical data. One of the key advantages of feedforward neural networks is their

simplicity and flexibility, as they can be applied to a wide range of tasks and data formats without explicit assumptions about the structure of the input data. They are often used as a baseline for comparison with more complex models.

However, feedforward neural networks are not suitable for processing grid-like data such as images or voxels. They ignore the spatial relationships between the input data and treat it as a flat vector of numbers, unable to effectively extract spatial features from the data. For example, processing an image of a cat with a feedforward neural network would result in a poor prediction due to the lack of spatial information [89].

To address this issue, *Convolutional Neural Networks* (*CNNs*) were developed to specifically process grid-like data such as images. *CNNs* use convolutional layers to extract features from the input data and then pass them through pooling layers to reduce the dimensionality of the data. This allows *CNNs* to learn hierarchical representations of the data and be invariant to translation, making them well-suited for tasks such as image classification and object detection.

However, with the advent of *Vision Transformers* (*ViT*) [70], there was a paradigm shift in the field of computer vision. Vision Transformers are based on the Transformer architecture [195], which was initially designed for natural language processing tasks. The *ViT* model divides an image into patches and then feeds them into a Transformer encoder, which captures the global context of the image. This enables *ViT* to process images without the use of convolutional layers and to outperform *CNNs* on most vision tasks [192, 206, 94, 141, 69].

It is important to note that the neural networks discussed in this text are based on *CNNs* without transformers. Transformer architectures are mentioned here only for the sake of completeness. The focus of the next section is on the differences between feedforward neural networks and *CNNs* in processing grid-like data, excluding transformer architectures.

2.2.2 Convolutional Neural Networks

Convolutional neural networks (*CNNs*) are a type of neural network that are specifically designed to process data that has a grid-like structure, such as an image or a voxel grid. A *CNN* uses a set of filters to extract features from the input data. These filters are convolved with the input data to produce a set of feature maps [89].

In a *CNN*, a feature map is a representation of the input data after it has been processed by a convolutional layer. Each element, or "neuron," in the feature map corresponds to a specific region in the input data, and the values in the feature map represent the presence or absence of certain features in that region.

For example, if the input data is an image, a feature map might represent the presence of edges, corners, or other patterns in the image. The filters used in the convolutional layer are designed to detect these features, and the resulting feature map is a representation of where in the input image these features are present.

The process of generating a feature map from the input data is called convolution. It involves applying a filter which is a small matrix of numbers, to the input data. The filter is slid across the input data, and at each location, the values in the filter are multiplied by the values in the input data at that location, and the results are summed to produce a single value in the feature map. This process is repeated for every location in the input data to produce the complete feature map. Feature maps are an important part of a *CNN* because they capture the most important features of the input data which can then be used by the rest of the network to make accurate predictions or decisions.

Convolutional two-dimensional filters are typically small, square matrices of weights that are learned during the training process. For example, a convolutional filter might be a 3x3 matrix of weights, as shown below:

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \tag{2.26}$$

To convolve the filter with the input data, the filter is slid across the input data, one element at a time, and the dot product is computed between the filter weights and the input data at each position. For example, if the input data is a two-dimensional matrix, the dot product would be computed as follows:

$$\hat{y}_{i,j} = \sum_{k=0}^{K-1} \sum_{l=0}^{L-1} x_{i+k,j+l} \cdot w_{k,l}, \tag{2.27}$$

where i and j are the row and column indices of the input data x, and K and L are the size of the filter.

The resulting feature maps are often passed through pooling layers. They are used to reduce the dimensionality of the data by summarizing the output of the convolutional layers. This has several benefits, including reducing the computational complexity of the network, and making the network more invariant to small translations of the features [89].

There are several different types of pooling layers, but the most common types are *max pooling* and *average pooling*. In a max pooling layer, the output of the convolutional layer is divided into a set of regions, and the maximum value from each region is selected and output as a new feature map. For example, if the input data is a two-dimensional matrix, the max pooling operation would be performed as follows:

$$\hat{y}_{i,j} = \max_{k=0}^{K-1} \max_{l=0}^{L-1} x_{i+k,j+l} \tag{2.28}$$

where i and j are the row and column indices of the input data, x is the input data, K and L are the size of the pooling window, and $\hat{y}$ is the output of the pooling layer. The max function is applied to, for example, each 2×2 region of the input data, and the resulting maximum value is used to generate the output feature map. This operation reduces the dimensionality of the data to a quarter, because the output feature map has half the number of rows and columns as the input data. In average pooling, the process is very similar but the average value from each region of the input data is used instead. Pooling is useful for *CNNs* because it reduces the size of the input data which makes the network faster and more efficient. Additionally, pooling allows the network to retain the most important features of the input data which are often distributed across multiple feature maps

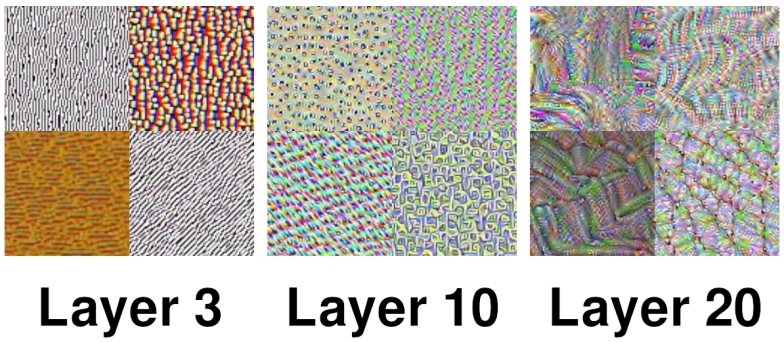

Figure 2.5.: **Increasing Complexity of the Features *CNN* Layers are Able to Capture.** The hierarchical structure of a *Convolutional Neural Network* (*CNN*) enables it to learn progressively complex features from the input data, starting with simple edges and corners in the first convolutional layers and building up a rich representation to make accurate predictions or decisions. Images are created using a *VGG-16* network [180]. A subset of the feature maps is shown. These activations are projected down to pixel space using a deconvolutional network approach.

and multiple convolutional layers. This can help the network to make more accurate predictions or decisions [89].

The process of applying convolutional filters and pooling the results is usually repeated multiple times and arranged in a hierarchical manner. By repeating the convolution and pooling process multiple times, a *CNN* can learn increasingly complex features of the input data. For example, in an image recognition task, the first convolutional layer might learn to detect simple edges and corners, the second convolutional layer might learn to detect more complex shapes, and so on. This hierarchical structure allows the *CNN* to build up a rich representation of the input data and make more accurate predictions or decisions. This hierarchical structure of filters is shown in Figure 2.5 with images created using an open source repository for the visualization of activations[1].

The final layers of a *CNN* depends on the specific task that the network is being used for. In general, however, a *CNN* will have one or more fully-connected layers at the end of the network. Fully-connected layers, also known as dense layers, are traditional feedforward neural network layers in which each neuron in one layer is connected to every neuron in the next layer. This allows the network to combine the information from all of the previous layers and make a final prediction or decision based on the input data.

In image recognition, the final fully-connected layers of a *CNN* can use the output of earlier convolutional and pooling layers to classify the input image into predefined categories. In an object detection task, the final layers of a *CNN* might be used to generate a set of bounding boxes around objects in the input image and to classify each object into one of several predefined categories.

Using a *CNN* for segmentation usually involves extracting features from an input image and reducing the resolution of the feature maps at the *bottleneck* layer. These feature maps can then be up-sampled to the input resolution using *transposed convolutions*. *Transposed convolutions* are essentially the opposite of standard convolutions: they up-sample the input feature map by inserting zeros between the elements and applying a convolutional filter. After up-sampling the feature maps to the input size,

[1] https://github.com/utkuozbulak/pytorch-cnn-visualizations

a final activation function, such as a *softmax*, can be applied to obtain class probabilities for each pixel in the input image. The class with the highest probability can then be selected as the predicted label for each pixel to obtain the final image segmentation.

2.2.3 Segmentation

Segmentation is the process of dividing an image into multiple segments or regions, each of which corresponds to a specific object or class of objects. This is different from classification which involves assigning a label to an entire image, rather than dividing it into multiple segments.

There are several different types of segmentation, including instance, semantic, and panoptic segmentation. Instance segmentation involves dividing an image into segments, where each segment corresponds to a different instance of an object. I.e., multiple segments may correspond to the same object class, but each segment corresponds to a different individual object within the image. Semantic segmentation involves dividing an image into segments, where each segment corresponds to a different object class. I.e., all segments of the same class are grouped together, regardless of whether they correspond to the same individual object or not. Panoptic segmentation is a combination of instance and semantic segmentation, where an image is divided into segments, each of which corresponds to a different object class, and individual instances of objects are also identified and segmented. This allows for a more comprehensive understanding of the objects and their relationships within the image.

Convolutional neural networks (*CNNs*) are commonly used for image segmentation tasks, as they are able to learn spatial hierarchies of features and can be used to effectively segment images into different regions or objects. There are different approaches to implementing *CNNs* for segmentation, including using fully convolutional networks [144] or encoder-decoder architectures [52]. While the examples provided in this section are related to images for ease of understanding and similarity to human vision, the concepts apply to the segmentation of three-dimensional point clouds in the same way as they do for images.

Instance Segmentation

Instance segmentation is a type of segmentation that involves identifying and segmenting each individual instance of an object in an image or a point cloud. For example, if an image contains multiple cars, instance segmentation would identify and segment each car separately, rather than treating all the cars as a single class.

A commonly used metric for instance segmentation is the *Intersection over Union* (*IoU*) metric. This metric measures the overlap between the predicted segmentation mask and the ground truth mask for a given instance. The *IoU* is calculated as the ratio of the area of overlap between the two masks to the area of union between the two masks. The mathematical equation for the *IoU* between two masks is

$$IoU = \frac{A_I}{A_U} = \frac{A_{pred} \cap A_{gt}}{A_{pred} \cup A_{gt}} \tag{2.29}$$

where A_I is the intersection, the area of overlap between the predicted and ground truth masks, A_U is the Union, the area of union between the two masks, A_{pred} is the predicted mask, and A_{gt} is the ground truth mask.

To evaluate the performance of an instance segmentation model, the *IoU* for each object in an image is typically calculated and then averaged across all instances in the image. This produces a single scalar value that can be used to compare the performance of different models

$$IoU_\mu = \frac{1}{N}\sum_{n=1}^{N} IoU_n \tag{2.30}$$

where N is the number of instances in the image, IoU_n is the *IoU* for the n-th instance.

In general, instance segmentation involves separating individual instances within an image or point cloud. Some definitions consider instance segmentation as class-agnostic separation of objects, while others require classification of the instances to distinguish foreground objects from the background [221, 138].

Semantic Segmentation

Semantic segmentation is a type of segmentation that involves assigning a semantic label to each pixel in an image or point in a point cloud. For example, semantic segmentation might classify the pixels of an image into classes such as cars, road, buildings, and trees. Unlike instance segmentation, semantic segmentation does not distinguish between individual instances of the same class.

The *Intersection over Union* (*IoU*) is also the metric of choice for semantic segmentation, more specifically the *mean Intersection over Union* (*mIoU*) metric. This metric measures the average overlap between the predicted segmentation masks and the ground truth masks for all classes in an image. The mathematical equation for *mIoU* is

$$mIoU = \frac{1}{C}\sum_{c=1}^{C} IoU_c = \frac{1}{C}\sum_{c=1}^{C} \frac{A_{I,c}}{A_{U,c}} = \frac{1}{C}\sum_{c=1}^{C} \frac{A_{pred,c} \cap A_{gt,c}}{A_{pred,c} \cup A_{gt,c}}, \tag{2.31}$$

where C is the number of classes in the image, IoU_c is the IoU for the c-th class, $A_{I,c}$ is the intersection, the area of overlap between the predicted and ground truth masks for the c-th class, $A_{U,c}$ is the area of union between the two masks for the c-th class, $A_{pred,c}$ is the predicted mask for the c-th class, and $A_{gt,c}$ is the ground truth mask for the c-th class.

The *mIoU* metric is useful for evaluating the overall performance of a semantic segmentation model on an entire dataset, as it provides a single scalar value that summarizes the model's performance across all classes in all images in the dataset.

Panoptic Segmentation

Panoptic segmentation is a type of segmentation which combines the capabilities of instance and semantic segmentation to provide a more complete and detailed understanding of a scene. It involves

identifying and segmenting each individual object instance, as well as assigning a semantic label to each pixel in the image or point in a point cloud. This allows for a more comprehensive understanding of the objects and their relationships within the image.

The *Panoptic Quality* (*PQ*) metric [119] measures the average *Intersection over Union* (*IoU*) between ground truth $\mathcal{S}$ and predicted segmentation masks $\hat{\mathcal{S}}$ for true positive pixels. The *PQ* is calculated by dividing the above by the sum of true positive, false positive, and false negative segments (divided by two) for a given class c

$$PQ_c = \frac{\sum_{(\mathcal{S},\hat{\mathcal{S}})\in TP_c} IoU(\mathcal{S},\hat{\mathcal{S}})}{|TP_c| + \frac{1}{2}|FP_c| + \frac{1}{2}|FN_c|}, \tag{2.32}$$

where a true positive for the class c is defined as an $IoU > 0.5$. The class-wise PQ_c is averaged over all classes to get the final system *PQ*.

The *PQ* metric can be seen as the multiplication of the *Segmentation Quality* (*SQ*) and *Recognition Quality* (*RQ*) [158]. The *SQ* measures the overall quality of the predicted segmentation, and is calculated as the *average IoU* between the ground truth and predicted segmentation masks for all true positive pixels. The *RQ* measures the overall quality of the predicted classes for each pixel in the predicted segmentation, and is calculated as the ratio of the total number of true positive pixels to the total number of true positive pixels plus half the number of false positive and false negative pixels. These two metrics can be written as follows:

$$SQ_c = \frac{\sum_{(\mathcal{S},\hat{\mathcal{S}})\in TP_c} IoU(\mathcal{S},\hat{\mathcal{S}})}{|TP_c|} \tag{2.33}$$

$$RQ_c = \frac{|TP_c|}{|TP_c| + \frac{1}{2}|FP_c| + \frac{1}{2}|FN_c|} \tag{2.34}$$

The *PQ* metric is the average of the *SQ* and *RQ* metrics

$$PQ_c = SQ_c \times RQ_c, \tag{2.35}$$

by which the *PQ* metric provides a single, scalar value that represents the overall performance of the model on the panoptic segmentation task. A high *PQ* score indicates that the model is able to accurately predict both the correct classes and instance for each pixel in the image.

2.3 Deep Learning on Lidar Data

In Section 2.1 lidar sensors and lidar data has been outlined. Raw lidar data is typically generated by a lidar sensor as a stream of three-dimensional points, with each point representing the position of an object surface reflection in the environment. Each point in the point cloud is indexed by a triplet of values representing the x, y and z Cartesian coordinates of the point. The point cloud may also include additional attributes or features associated with each point, such as the intensity of the reflection.

This list of multi-dimensional entries is typically highly unstructured and irregular, with a large number of points distributed in three-dimensional space. This makes it difficult for deep neural networks which are designed to handle structured and regular input data, to effectively learn patterns and features from the data.

One of the first networks to successfully process raw point clouds was the *PointNet* [161] architecture. *PointNet* consists of a single fully-connected layer which takes as input a set of three-dimensional points and applies a series of transformations to extract features from the points. The extracted features are then fed through a series of fully-connected layers which are used to classify the points. *PointNet* has been widely adopted in a variety of applications, and has demonstrated strong performance on a number of benchmarks for point cloud processing tasks. The simplicity and flexibility of the architecture has made it a popular choice for researchers and practitioners working with point clouds.

PointNet++ [162] was an extension of *PointNet*. *PointNet*++ was designed to address some of the limitations of *PointNet*, such as its reliance on global context and its inability to capture fine-grained features and patterns in the point cloud. *PointNet*++ uses a hierarchical structure to process the point cloud which allows it to capture both global and local context in the point cloud. At each level of the hierarchy, *PointNet*++ uses a sampling and grouping operation to select a set of representative points, and applies a *PointNet*-like architecture to these points to extract features and predict properties of the points. The features and predictions from each level of the hierarchy are then aggregated to form a global representation of the point cloud.

Based on this hierarchical approach of *PointNet*++ as well as the advances of *CNNs*, three-dimensional voxel based networks [230, 188, 228, 10] improved the performance on lidar data in deep neural networks immensely. Point cloud voxel nets are designed to handle large and complex point clouds by dividing the point cloud into a set of voxels which are cubic cells in three-dimensional space. Each voxel is assigned a set of points from the point cloud, and a *PointNet*-like architecture is applied to each voxel to extract features and predict properties of the points. The same concept can also be applied to two-dimensional birds-eye-view, top-down representations of the point cloud [127, 232]. The grid structure of the voxels enable the use of convolutional operations in the two-dimensional and three-dimensional grid patterns of the voxels. Encoder-decoder structures similar to that used in image segmentation improve semantic segmentation, object detection and other tasks on the lidar point clouds compared to the *PointNet* based networks.

Another pre-processing method for using unstructured and irregular lidar data with deep neural networks is the range image projection [211, 7]. A range image is a two-dimensional grid of cells that represents the distance from the sensor to each point in the environment. Each cell in the range image is assigned a value that represents the distance to the nearest point in the environment, and the values are typically represented as a set of discrete range bins.

Range image projections are often used in lidar-based perception systems, as they provide a compact and efficient representation of the sparse and unevenly distributed point cloud that is easy to process and analyze. Range image projections can also be used to visualize the point cloud and to understand the structure and geometry of the point cloud as shown in Figure 2.6.

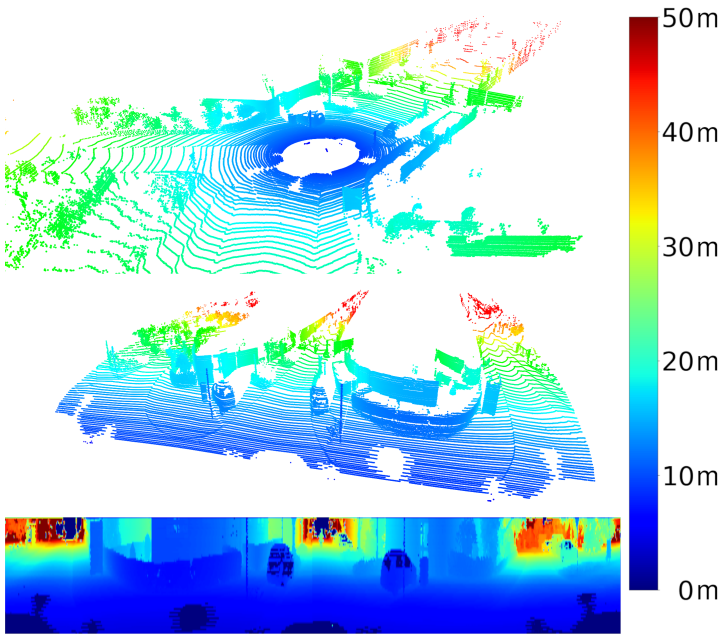

Figure 2.6.: **Range Projection of a Three-Dimensional Point Cloud.** The Cartesian point cloud (top) is first projected onto the spherical coordinate system (middle) and then collapsed along the range dimension to generate the final range image projection (bottom). The color of each point in the image corresponds to its distance from the lidar sensor, with blue indicating points that are close and red indicating points that are far away.

The latter pre-processing method is used multiple times throughout this thesis, as the lower dimensional representation shows the three-dimensional point cloud from the point of view of the lidar sensor. Range image projections are closely related to the raw data that is output by the lidar sensor which makes them well suited for developing sensor-centric methods. In Section 2.1 the working principle of a single lidar module was presented, that yields only a depth value for a given one-dimensional laser beam. The range image is therefore much closer to the original data. By using a range image projection, it is possible to develop methods that are closely tied to the sensor data and that are optimized for the specific characteristics and limitations of the sensor.

The range image projection typically necessitates the transformation of a Cartesian point cloud into a spherical coordinate system. This transformation maps each Cartesian point $(x, y, z \in \mathbb{R})$ to the spherical coordinates $(r \in \mathbb{R}^+; \phi, \theta \in \mathbb{R})$:

$$\begin{pmatrix} r \\ \phi \\ \theta \end{pmatrix} = \begin{pmatrix} \sqrt{x^2+y^2+z^2} \\ atan2(y,x) \\ asin(\frac{z}{r}) \end{pmatrix}, \text{ with } r \in [0, \text{inf}) \text{ and } \phi, \theta \in [-\pi, \pi], \tag{2.36}$$

where each point is projected from the coordinates x, y, z to the sphere coordinates r, ϕ, θ, which represent the range, azimuth angle, and elevation angle, respectively.

When transforming Cartesian coordinates to spherical coordinates, particularly in the context of automotive lidar sensor data, it's crucial to adopt a convention that closely aligns with the intuitive understanding of the Cartesian system.

The inclination angle θ is often defined in physics and mathematics as the angle from the positive z-axis to the point in question, using the $acos$ function. This convention can be counter-intuitive in the context of automotive lidar sensors. Here, the angle of interest is the one ascending from the xy-plane, which is usually aligned with the ground plane.

To maintain proximity to the Cartesian coordinate system and to align with the intuitive understanding of angles in the context of lidar sensors, θ is defined as the angle from the xy-plane to the point in question. The $asin$ function is used to calculate the defined inclination angle.

Using the azimuth and elevation angle, pixel coordinates are defined in the horizontal and vertical image directions u and v, thus mapping the points from $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ by

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{1}{2}[1 - \phi\pi^{-1}]w \\ [1 - (\theta + f_{up})f^{-1}]h \end{pmatrix}, \text{ with } u, v \in \mathbb{N}_+, \tag{2.37}$$

where (u, v) are the image pixel coordinates, (h, w) are the height and width of the resulting range image and $f = f_{up} + f_{down}$ is the vertical field-of-view of the lidar sensor. The image coordinates (u, v) retain the depth information, thereby preserving the complete three-dimensional information in a two-dimensional representation.

The principle of coordinate transformations to the spherical coordinate system and the range image domain is consistently employed throughout this thesis in diverse contexts. The fundamental premise is that a three-dimensional point cloud can be transposed into a two-dimensional image in the range image domain, generally without any loss of information. However, careful consideration is required for pixel coordinates that do not have an associated point in the three-dimensional cloud. Such discrepancies may arise from factors like light absorption, total reflection, or objects being beyond the sensor's range. In these cases, the affected pixels are often explicitly flagged as 'not available' or alternatively assigned a zero or maximum range value. This treatment varies based on the specific requirements of the application. In this work, the latter approach is consistently adopted for handling such instances.

The representation of a point cloud as a two-dimensional image facilitates the application of image processing techniques for data analysis and manipulation. This approach streamlines the analysis process and enables the utilization of sophisticated tools, originally developed for image processing, on point cloud data.

Part I

Developing Novel Algorithms and Networks for Lidar Segmentation

Contributions to Instance Segmentation: 3

Developing a New Clustering Algorithm for Lidar Data

As an introduction to segmentation, and to establish the sensor-centric approach that accompanies this work throughout, instance segmentation is presented in the form of point cloud clustering algorithms.

In this chapter, an algorithm that has already been published in a conference paper [99] is described in detail. Sections from the mentioned paper are presented here in their entirety or paraphrased. Additionally, an initial version of the method was published in the author's master's thesis [96].

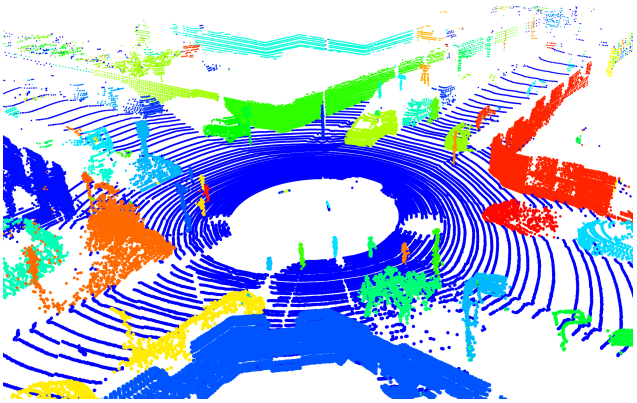

Figure 3.1.: **Instance Segmentation Results of the Novel Method Outlined in this Chapter.** A three-dimensional point cloud is shown with clustered points. Every instance is assigned a random color.

The "Fast Lidar Image Clustering" (*FLIC*) algorithm achieves highly accurate point cloud segmentation and can run in real time, processing data at a rate multiple times faster than the sensor's recording frequency with minimal fluctuation, regardless of the scene's context. This is accomplished by working directly on the laser range values of the sensor represented as a range image.

The *FLIC* algorithm overcomes the problem of sparsity in the point cloud by enforcing a two-dimensional neighborhood on each measurement, allowing it to work with dense, two-dimensional data with clearly defined neighborhood relationships between adjacent measurements. A Python implementation of the algorithm runs in real time on a single Intel® Core™: i7-6820HQ CPU @ 2.70 GHz core, achieving a frame rate of up to 165 Hz. An example of the lidar instance segmentation produced by the algorithm is shown in Figure 3.1.

The following main contributions are provided in this chapter:

- A novel, real-time capable, CPU-based lidar range image clustering algorithm.
- A method to reduce the under-segmentation of lidar clusters.
- An evaluation of the proposed system on multiple open source lidar datasets.
- A comparison of the algorithm with state-of-the-art clustering methods for lidar point clouds.

3.1 Related Work

Instance segmentation of lidar data is possible via various methods, such as using machine learning algorithms to separate objects of unknown classes from each other by learning on labeled data [126, 223, 217]. Other approaches use heuristic methods which are hand-designed algorithms that use domain-specific knowledge to process the data and separate instances from each other [149, 45, 220]. These methods can be used individually or in combination with the former, where the output of the heuristic methods are fed into machine learning algorithms to improve their performance [153, 132].

The main disadvantage of heuristic methods is that they can be difficult to design and require a high level of expertise in the domain. Additionally, heuristic methods are not as flexible as machine learning algorithms which can adapt to new data and changing conditions. Finally, heuristic methods may not be able to generalize to new situations, whereas machine learning algorithms can learn to do so with enough training data.

On the other hand, the main advantage of heuristic methods is that they are hand-designed algorithms that are based on domain-specific knowledge and experience. This means that they can be tailored to the specific problem at hand and can often produce good results without the need for extensive training data. Additionally, heuristic methods are usually computationally very efficient as they are precisely adapted to the application [45, 227].

3.1.1 Clustering Algorithms

Clustering is a technique used in data mining and machine learning to group a set of data points into clusters based on their similarities and dissimilarities. This is typically done using a distance measure, such as Euclidean distance, to calculate the similarity between data points [41]. Clustering is useful for various applications, including pattern recognition, data compression, and anomaly detection.

There is a significant body of research on lidar clustering, particularly in automotive applications. Most approaches focus on improving both segmentation accuracy and execution time. Many of these methods involve separating objects in three-dimensional space, resulting in high accuracy but long runtimes. Notable examples of three-dimensional lidar clustering algorithms include *DBSCAN* [77], *Mean Shift* [84, 59], and *OPTICS* [32].

Other approaches to lidar clustering include voxelization to reduce the point cloud's complexity and find clusters in the resulting representation [104] or using bird's eye view projection coupled with height information to separate overlapping objects [123].

The authors of [149] have proposed using local convexity criteria on the spanned surface of three-dimensional lidar points in a graph-based approach. The authors of [45] used a similar criterion - the spanned angle between adjacent lidar measurements relative to the lidar sensor origin - to segment objects. They further utilized the neighborhood conditions in the range image to achieve the fastest execution time to date. The authors of [220] exploited the sequential relationship in scan

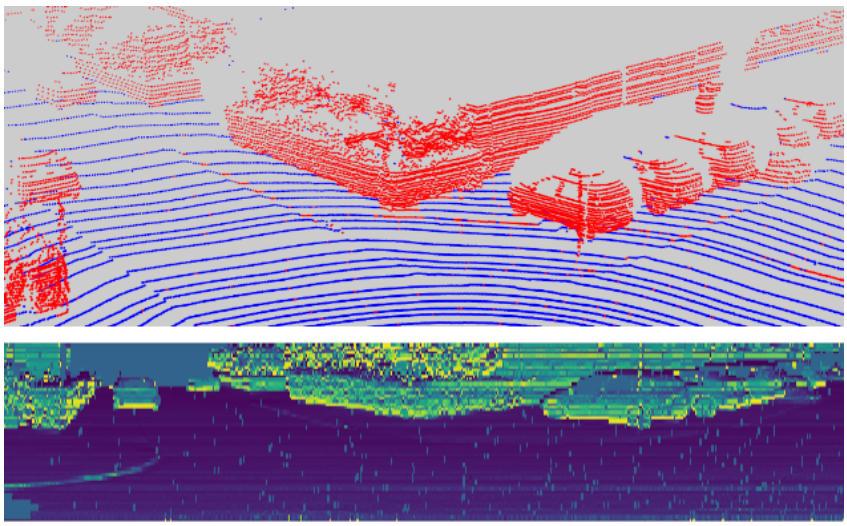

Figure 3.2.: **Visualization of the Ground Segmentation Method of *FLIC*.** *Top*: Ground segmentation; Blue points represent ground points, red points are not part of the ground. *Bottom*: Angle image used to find horizontal surfaces for the ground segmentation.

lines of lidar sensors to find break points in each line and merge the separate lines of channels into three-dimensional objects in a subsequent step.

Other methods use machine learning directly on three-dimensional point clouds [126, 223, 217], projections into a camera image [205], or on spherical projections of lidar points in the range image space [207], to segment object instances in point clouds. While these methods show promise in some cases, their longer runtimes currently prevent their application on embedded automotive hardware.

3.2 Fast Lidar Image Clustering: Method

The raw data from lidar sensors is typically provided in one of two formats: either as a three-dimensional representation of the measurements or as a list of range measurements. Each range measurement is coupled with a number that relates to the lateral position and channel, such as the index of the lidar module used to collect the data. These two values correspond to the y- and x-position of the measurements in the range image. The two representations can be converted to each other via a sphere projection applied to the three-dimensional data or a Cartesian projection applied to the spherical values.

Regardless of the data's origin, whether it came directly from the range measurements of the sensor or from a three-dimensional point cloud, the data is further processed in the range image format. It undergoes the same two-step process of ground extraction and subsequent clustering of contiguous points.

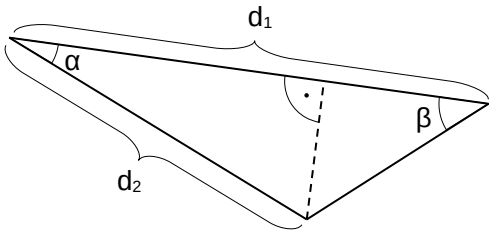

Figure 3.3.: **Illustration of a General Triangle.** The angle β is unknown, but can be calculated using the lengths of the sides d_1 and d_2, along with their corresponding angle α. By applying the trigonometric relationship, specifically the arctan function, β can be calculated using Equation 3.1

3.2.1 Ground Extraction

For object segmentation, it is assumed that the lidar is mounted on a ground-based vehicle. To prevent the algorithm from erroneously connecting two instances through the ground plane, the points belonging to the ground plane are removed from the main segmentation process. However, a simple height-based threshold is not sufficient due to the unevenness of the road surface. Furthermore, the orientation of the vehicle, such as pitching and rolling, can also influence the way the ground is perceived in the sensor data, adding to the complexity of the segmentation process.

Given the detailed specifications of the lidar sensor, the angular position data for each channel is utilized to ascertain the incident angle of the laser beam upon a surface. This information enables the exclusion of range image values that correspond to a horizontal plane beneath a specified height threshold. To achieve this, each cell of the range image is compared with its neighboring cell above it to calculate the angle spanned between the line connecting the two and a line from the sensor's location to the cell in question

$$\beta = \arctan\left(\frac{d_2 \cdot \sin\alpha}{d_1 - d_2 \cdot \cos\alpha}\right), \tag{3.1}$$

in which the selected cell value is d_2, the one above the cell is d_1 corresponding to the respective depth measurement and α is the angle between the two measurements.

The trigonometric relationship between the depth values d_1 and d_2 as well as the angles α and β can be seen in Figure 3.3.

As a result, the image shown in Figure 3.2 is obtained, representing the angle values of the lidar beam in relation to the connection spanned between the current lidar measurement and the lidar channel below, as shown in Figure 3.4.

The β value in itself does not hold any meaning in relation to the ground plane. Therefore, the relative angle is subtracted from the mounting angle of the given lidar channel in relation to the horizontal view plane δ.

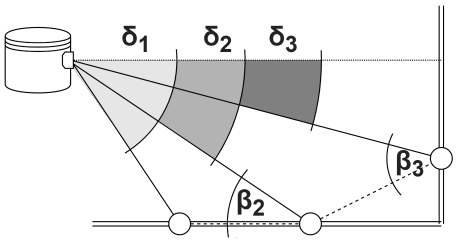

Figure 3.4.: **Schematic Visualization of the Geometric Properties for the Ground Angle Determination.** The angle measurement of lidar points in the vertical direction is used to define a horizontal orientation for the ground plane extraction. (Note that there is not β_1, as the first lidar channel has no previous channel, β_2 is therefore extended to the first channel.)

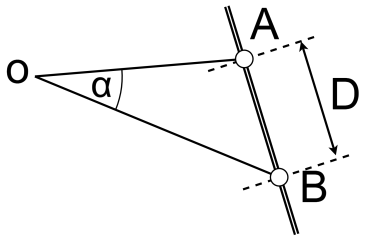

Figure 3.5.: **Schematic Visualization of the Geometric Properties for the Point-Wise Distance Calculation.** With the lidar Sensor in O, the lines OA and OB show two neighboring distance measurements. The distance between the two measurements is calculated using the spanned angle α between the points.

Using a lookup table of the absolute channel angles δ_r with respect to the channel r, all range image cells that span a horizontal up to a certain threshold angle θ of $\pm 10^\circ$ to the sensor horizon are classified as ground points

$$-\theta < \delta_r - \beta_r < \theta, \tag{3.2}$$

where r corresponds to the given channel index.

This method classifies all lidar measurements reflected by horizontal surfaces. To prevent excessive removal of valid measurements from elevated horizontal surfaces such as car roofs or hoods, a height image is used, in which the range values are replaced by the Cartesian z-coordinate in relation to the ego vehicle. This image is used to keep all horizontal surfaces above a certain height: a line from the ground position of the ego vehicle to the maximum possible elevation spanned by the 10° slope threshold. A comparable metric was described in [58] although with relative height thresholds instead of absolute ones. The decision fell on absolute values to reduce the computation time. Figure 3.4 depicts the relationship of the channel angles δ_r (angle in relation to the horizontal 0° line) to the surface angles β_r.

3.2.2 Clustering

A heuristic approach is employed to perform object instance segmentation on lidar sensor data by clustering three-dimensional points in the two-dimensional range image space. The *FLIC* algorithm relies on the removal of the lidar measurements belonging to the ground plane from the range image.

Inspired by the work in [45], the neighborhood relationship of adjacent measurements in the range image is used for an efficient clustering. As visualized in Figure 3.5 the given range values $\|OA\|$ and $\|OB\|$ are compared for each pair of neighboring lidar measurements. The cosine law is applied to calculate the Euclidean distance D in the three-dimensional space using the two-dimensional range image with

$$D = \sqrt{\|OA\|^2 + \|OB\|^2 - 2\|OA\| \cdot \|OB\| \cos\alpha} \tag{3.3}$$

$$= \sqrt{d_1^2 + d_2^2 - 2 \cdot d_1 \cdot d_2 \cos\alpha}. \tag{3.4}$$

The α angle between adjacent lidar measurements is required for the calculation and is usually provided by the manufacturer of the lidar sensor for both the horizontal and vertical direction [198, 197, 196, 199, 103]. A threshold value is defined to determine if two points that are close enough together belong to the same object. Points that exceed this value are considered too far apart to be neighboring points on the same object. Typically, neighbors on a given object are relatively close to each other. The distances of two neighboring points in the range image from two separate objects are substantially larger. Pixel coordinates, that do not have a corresponding point in the three-dimensional point cloud are filled with the sensor's maximum range value to prevent undersegmentation, i.e, a connection of two objects via an empty pixel position.

The computational is reduced by exclusively using variables which are given by the range measurements and by pre-calculating the cosine of the given angle resolutions, the calculation of the squared Euclidean distance are reduced to a total of four scalar multiplications, an addition and one subtraction

$$D^2 = d_1 \cdot d_1 + d_2 \cdot d_2 - \underbrace{2 \cdot \cos\alpha}_{\text{Constant}} \cdot d_1 \cdot d_2. \tag{3.5}$$

With the defined distance threshold, all lidar points in the range image can be connected to separated clusters and cluster-less background points. The Euclidean distance as a threshold value enables a single parameter implementation with a clear physical meaning which is adaptable to different sensors.

A good performance was reached with a threshold of 0.8 m as the limit for a connection between two points. This threshold theoretically enables the clustering of three-dimensional objects with a Velodyne *HDL-64E* up to 114.6 m before the measured points are too far apart on a flat vertical surface. Horizontally connected components can, in theory, be detected up to a distance of 509.3 m which is more than four times the reliable range of the sensor for vehicles of 120 m as defined by the manufacturer [196].

Connected-Component Labeling

FLIC exploits the grid structure of the range image to repurpose operations commonly used in image processing. Specifically, the three-dimensional Euclidean clustering is transformed into a two-dimensional *connected-component labeling* (*CCL*) problem. To do so, two virtual copies of the range image are created: one copy is shifted by one pixel over the x-axis and the other over the y-axis. These shifted images are used to compare each range value in the original range image

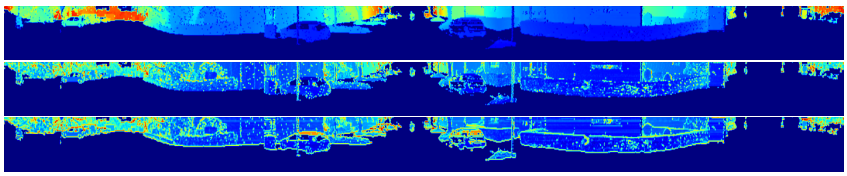

Figure 3.6.: **Range Image and Distance Images of the Three-Dimensional Distance Between Neighboring Lidar Points in the Range Image.** *Top:* range image. *Mid:* horizontal distance between each range value and its direct neighbor to the right in the original range image. *Bottom:* vertical distance between range values.

with its vertical and horizontal neighbor over the whole image. Thus Equation 3.5 calculates the three-dimensional distance on these images for each point with its vertical and horizontal neighboring measurement. Applying the threshold on the resulting distance values yields two binary images representing the connection or separation between two points in the range image. The original range image is furthermore reduced to a binary image representing the presence and absence of lidar measurements for all pixels of the range image. A visualization of the range image and the two distance images of an example scene of the KITTI dataset [86] can be seen in Figure 3.6.

The three created binary images contain all the required information to segment the lidar measurements of the whole frame into clusters and background points. For this, a simple and efficient image processing algorithm is used; *connected-component labeling*. The 4-connected pixel connectivity, also known as *von Neumann neighbourhood* [191], is defined as a two-dimensional square lattice composed of a central cell and its four adjacent cells. To apply the pixel connectivity to the present lidar data, the binary lidar presence values are combined with the binary threshold images of the distances between lidar points. By arranging these three images as shown in Figure 3.7, it is straight forward to apply *CCL* algorithms with a 4-connectivity on the resulting image to label each island of interconnected measurements as a different cluster as shown in Figure 3.8. The resulting segmented image is then subsampled to the original range image size. Thus the three-dimensional cluster labels are directly taken from the *connected-component image*, as each pixel corresponds to a given lidar point in the three-dimensional point cloud.

There are a multitude of CPU-based implementations for *CCL* problems most common are the "*one component at a time*" [2] and the *two-pass* algorithm [107]. The first method is in this work in the form of the straight-forward implementation of the *scipy* library "*label*" for n-dimensional images [201], as it provides a fast *cython* based function. More recent *CCL* algorithms make use of GPUs by applying the *CCL* in parallel [101, 29]. This is a very promising approach, as all previous processing steps in this work were applied to rasterized images and can be directly computed in parallel on a GPU (This was not attempted in this approach, as the main goal was a real-time application for CPU-based automotive hardware. A new and improved GPU implementation of the *FLIC* separation metric is outlined in Chapter 5.3).

In a subsequent step to the CPU-based *CCL*, a threshold on the labeled clusters is applied for objects below a certain number of lidar measurements to reduce false clusters resulting from noise in the sensor. In the evaluation of this chapter a minimum of 100 points was chosen to be considered a valid cluster candidate following the work in [45]. Objects of interest are cars, pedestrians and other

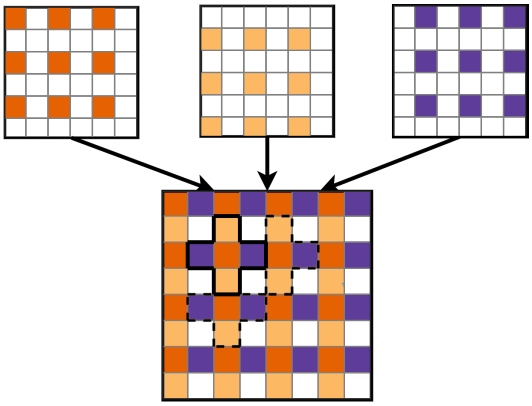

Figure 3.7.: **Schematic Visualization of the Combination of Binary Images to Connect Lidar Points.** The orange squares represent the binary value of present lidar measurements, the beige and purple squares represent the horizontal and vertical connections of these measurements respectively. The solid and dashed crosses show the 4-connectivity used by the *connected-component labeling*.

Figure 3.8.: ***Connected-Component Label* of a Car Resulting from the Combined Binary Images.** The binary range image and two binary neighbor connections are combined into a connection image as shown in FIGURE 3.7.

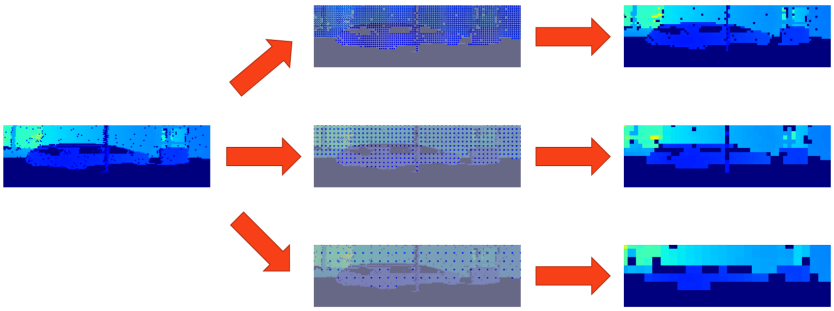

Figure 3.9.: **Range Image Sub-Selections for *Map Connections*.** The original range image is subsampled for every second (top), every forth (mid) and every eighth (bottom) value, to create sparser copies for the *Map Connections*.

road users in the close and mid range distance. Lower thresholds are recommended to include static objects such as poles and debris on the road as well as objects farther away from the sensor.

Thus the lidar measurements are segmented into *connected components* of separate objects and non-segmented points which corresponds to the ground plane and background noise.

Map Connections

Heuristic lidar segmentation algorithms are prone to under- and oversegmentation [149, 45]. Undersegmentation refers to the problem where multiple objects are grouped into a single segment. On the other hand, oversegmentation is when a single object is segmented into multiple smaller segments. These problems arise often in lidar clustering due to the characteristics of lidar sensors. Noise, occlusion, sparsity and missing measurements resulting from deflected laser beams lead to missing connections between areas of the same object [149, 45]. This causes the direct neighborhood approach described above to oversegment single objects into multiple clusters. Examples of such challenging instances are shown in Figure 3.10.

To overcome the limitations of the direct neighborhood approach and to ensure a more robust segmentation, *FLIC* is extended by what is called *Map Connections* (*MCs*) in the following. The schematic visualization in Figure 3.11 displays a connection of each measurement with its second neighbor. Due to the known α angle between all measurements, the Euclidean distance calculation can be extended from each measurement to any other in its vertical column or horizontal row while still using the cosine law described in Equation 3.5, by adjusting the angle to the given offset. This allows to robustly connect segments of the same object which have no direct connection due to missing measurements or obstruction by other objects in the range image. By sub-sampling the whole range image the resulting binary image for the *CCL* algorithm shrinks accordingly as shown in Figure 3.9.

An example of the improved segmentation can be seen in Figure 3.10. In the evaluations in Section 3.3 1, 6 and 14 *MCs* have been added along the main diagonal of the range image respectively. The 6 *MC* connections are shown in Figure 3.12. The *Maps* of reduced point-sets are smaller than the original point-set and thus require only a fraction of the computation time on top of the

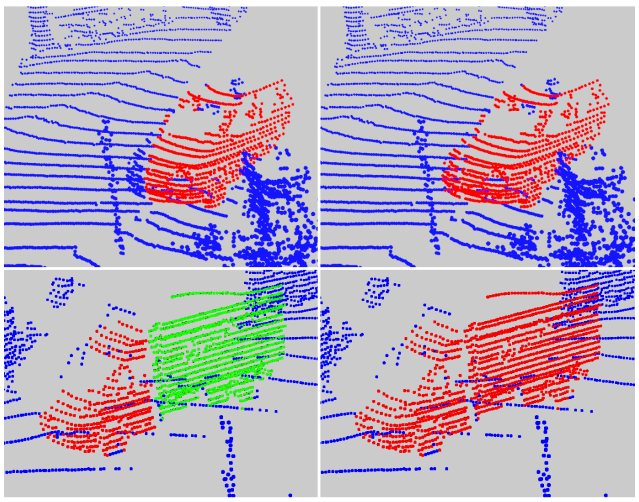

Figure 3.10.: ***Map Connections* Reduce Over-Segmentation.** Left: Results using only the direct connectivity between neighboring Lidar points. Right: A single additional *MC* between every second Lidar measurement. The proposed *MCs* enable a more accurate segmentation of the car (top) and reduce the over-segmentation of partially occluded objects, as shown by the truck in the bottom images.

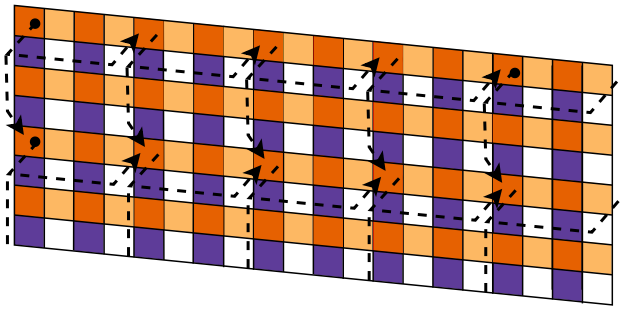

Figure 3.11.: **Schematic Visualization of *Map Connections*.** Additional *Map Connections* (dotted lines) between non-neighboring Lidar points on top of the direct connections to neighboring points (yellow and blue squares).

directly connected clusters. The additional mapping of the cluster-ids of the original clusters with the *MCs*, results in a slightly increased runtime as shown in Section 3.3.1. The combined use of the direct connectivity of neighboring measurements and the *MCs* enable a pseudo three-dimensional Euclidean clustering while exploiting the fast runtime of two-dimensional pixel connectivity. Thus, the quality of the segmentation improves without sacrificing the real-time capability of the method.

3.3 Fast Lidar Image Clustering: Evaluation

The evaluation of the presented instance segmentation algorithm via range image clustering addresses the two main goals of the application: speed and accuracy. The first experimental evaluation evaluated

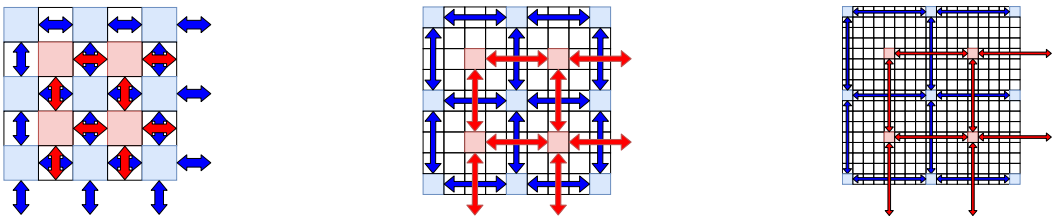

Figure 3.12.: **Pattern of Six *Map Connections*.** The pictured structure increases the connection area with the least amount of maps. The colored cells are pixel coordinates within the range image that are compared beyond their direct neighbors via the *Map Connections* that are shown as arrows.

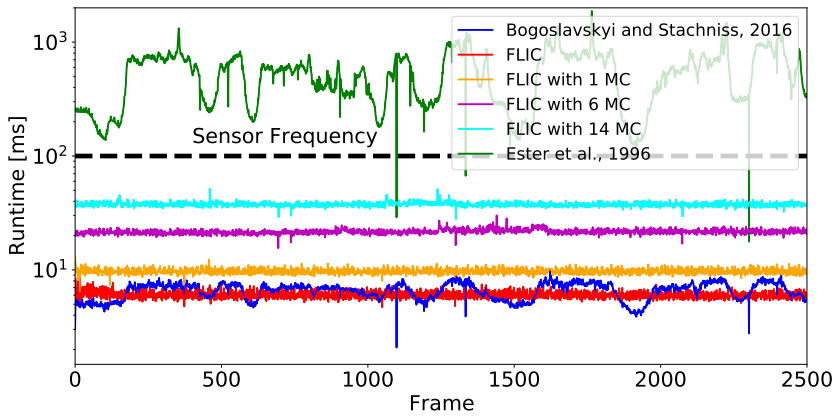

Figure 3.13.: **Frame-Wise Runtime of *FLIC* and Other Algorithms on a 64-beam Velodyne Dataset [148].** Please note the logarithmic scale for the runtime.

the method's ability to run in real time at typical sensor recording frequencies, preferably with a constant processing rate and minimal fluctuations regardless of the scene's context. The second experiment focused on a quantitative assessment of segmentation quality.

3.3.1 Runtime

Following the experimental setup in [45], the first experiment was carried out on the provided data by Moosmann et al. [148] to support the claim, that the proposed approach was capable of online segmentation in real time. All listed methods have been evaluated on the same Intel® Core™ i7-6820HQ CPU @ 2.70 GHz.

Figure 3.13 shows the execution time of the five methods on the 2,500 Frames dataset [148]. The *FLIC* algorithm ran with an average of 165 Hz and was therefore faster than the previously fastest published algorithm of [45] of 152 Hz, while also exhibiting less fluctuation due to the binary image implementation when used without any additional *MCs*. A box-plot of the average runtime of [45] and the *FLIC* can be seen in Figure 3.14 which shows the fluctuating nature of methods depending on the scene context, as opposed to the presented *FLIC*.

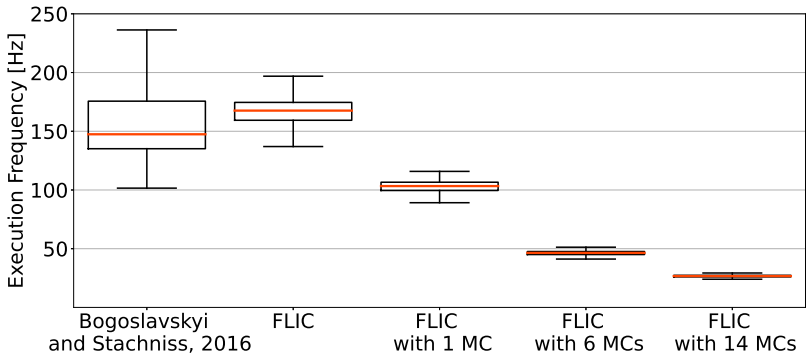

Figure 3.14.: Average Segmentation Frequency of 2,500 Scans from a 64-beam Velodyne Dataset [148]. The presented approach with up to 14 *Map Connections* is compared to the method by Bogoslavskyi and Stachniss [45].

As can be seen in Figure 3.13, when adding MCs to the proposed method, the execution time suffers from a slightly longer runtime, while still running at a frequency of 26 to 105 Hz depending on the number of additional *MCs*. This is 2.6 to 10.5 times faster than the recording frequency of the lidar sensor.

3.3.2 Segmentation Quality

For the evaluation of segmentation quality of the *FLIC* algorithm, the dataset *SemanticKITTI* [39] was used. This dataset enriches the *KITTI* datasets [86] odometry challenge with semantic and instance-wise labels for every lidar measurement.

To reduce the influence of the proposed ground plane extraction in Section 3.2.1 and focus on the results of the clustering mechanisms, the evaluation was conducted once without the lidar points of the classes *"road"*, *"parking"*, *"sidewalk"*, *"other-ground"*, *"lane-marking"* and *"terrain"*. And a second time without any usage of the semantic labels by applying the ground extraction proposed in Section 3.2.1 on all methods.

For each ground truth object with at least 100 lidar point measurements, each algorithm's object cluster output with the most ground truth overlap was selected. Using these two lists of points, the *Average Intersection over Union* (IoU_μ) was calculated by averaging the *Intersection over Union* values of every single instance over 10 sequences.

The connection and separation of instances solely through the distance carried the risk of under-segmentation, e.g., in the case of objects that are very close together. For this reason, the results were measured instance-wise in the evaluation. If two instances were represented by only one cluster, only the object with the higher *IoU* was counted, while the second object was marked as "not found". This metric was computed for each algorithm listed in Table 3.1, for ten annotated sequences with lidar instance ground truth in the dataset. The quality of *FLIC* was directly compared to the, at the time, fastest lidar clustering algorithm [45], as well as the very precise three-dimensional Euclidean density clustering algorithm *DBSCAN* [77]. The *scikit-learn's* [156] implementation of the *DBSCAN*

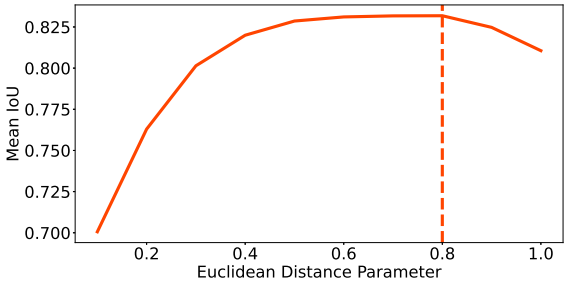

Figure 3.15.: **Parameter Study of the Maximum Distance Between Two Points, to be Considered Part of the Same Cluster.** The dashed line shows the maximum IoU for the evaluation log. The plateau between 0.5m and 0.8m shows a very broad and robust sweet spot for the *FLIC* algorithm.

algorithm was used in these evaluations. Considering the age of this algorithm, it might be surprising to see that it is still used in modern clustering applications [224, 145, 55]. This long-term relevance was also confirmed by the "*Test of Time*" Award from *ACM SIGKDD* [179]. The algorithm was also revisited by the original authors in a follow-up paper [175] in 2017 to demonstrate the continued relevance in many clustering applications. Therefore the *DBSCAN* was used to compare the pseudo three-dimensional approach of the *FLIC* algorithm to two state-of-the-art algorithms, one for speed and one for accuracy.

The IoU_μ with the best-performing parameters of each method is presented in Table 3.1. With the threshold parameter set as 0.8m, the FLIC *algorithm* outperformed [45] with only the direct neighborhood implementation without any *MCs* while also exhibiting a faster run-time. This parameter has been set with an additional experiment on a single log of the dataset, which was excluded from the other validations. The results of this experiment are shown in Figure 3.15. The Euclidean distance threshold has a clear physical meaning, that directly defines the connection of points. It has a large global optimum and can be fine-tuned to different point densities of various lidar sensors.

On average, the *FLIC* algorithm is 8% faster than the algorithm presented in [45]. However, for the longest execution time of a given frame, there is a 15% difference between the two algorithms. Together with the proposed *MCs* between all odd measurements, *FLIC* performs with an IoU_μ score even higher than the *DBSCAN* [77], as shown in Table 3.1 while *FLIC* is faster by a factor of 120.

Using six *MCs*, *FLIC* surpassed the performance of *DBSCAN* with an even larger margin and managed to reach a noticeably higher IoU_μ. A total of 14 *MCs* outperformed the three-dimensional *DBSCAN* algorithm on four of the six shown metrics in Table 3.1, with an average execution frequency of 26 Hz it still runs at 2.6 times the sensor frequency.

Without any *MCs FLIC* is on average 120 times faster than the *DBSCAN*. With an increasing number of additional *MCs* it is 67, 25 and 14 times faster than the *DBSCAN* Algorithm. The run-time increase did not scale logarithmic as one would expect with additional *MCs* (as they re-apply the same function to a smaller subset of the original point cloud). This issue resulted from the computational overhead caused by the python implementation for the cluster matching between the maps. The algorithm

was not re-implemented in a different programming language, as the computation time was still far below the sensor frequency.

The second column of Table 3.1 shows, that the proposed ground extraction method of Section 3.2.1 degraded the performance of all listed algorithms, except for [45]. This results from the fact, that the metric for the ground segmentation is very similar to the cluster separation metric used by Bogoslavskyi et al. [45]. A better ground separation leads to a much better performance for the proposed method as the IoU_μ values with the ground truth (GT) ground segmentation shows. Improving the ground separation is therefore critical to improve the instance segmentation of the *FLIC*.

For further evaluation of the instance-level performance, the object segmentation was computed similarly to [45] by calculating the recall, which is the fraction of true positive detections out of all GT instances. Ten bins of point-wise overlap of the ground truth and segmented clusters were defined, ranging from an *IoU* of 0.5 to 0.95 in steps of 0.05. The recall of all bins was averaged into one single metric score, the *average Recall* (R_μ), which is shown in Table 3.1 for each method. In addition to the overall precision metric, the evaluation also reports the recall for different overlap values (0.5, 0.75, and 0.95) where an object is considered correctly segmented if its *IoU* with the ground truth is greater than the respective overlap threshold. The recall, that showed how many instances are matched with an *IoU* of more than the threshold *thr* is therefore defined as

$$\begin{aligned} R_{thr} &= \frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M} a_{n,m} \\ \text{with } a_{n,m} &= \begin{cases} 1, \text{ if } IoU(n,m) >= thr, \\ 0, \text{ else.} \end{cases} \end{aligned} \tag{3.6}$$

for N instances and M clusters in which each instance and cluster was matched via the *Jaccard Index* (*IoU*) over a certain threshold *thr*. Please note, that due to the definition of the *Intersection over Union* only one cluster can match a ground truth instance with an $IoU > 0.5$.

Table 3.1 shows, that the *FLIC* algorithm matched on average more GT instances than [45] and are close to the mean segmentation recall of the *DBSCAN* [77] algorithm. With a higher number of *MCs*, *FLIC* achieved better recall values for overlap values of 0.5 and 0.75, while the *DBSCAN* [77] algorithm matched more instances with higher overlap values due to the full three-dimensional clustering on all points of the dataset.

Only up to 14 *MCs* were compared in this evaluation, in order to preserve the real-time capability. However, with just these 14 *MCs*, *FLIC* achieved a clustering segmentation which performed comparable to, and in some regards better, than the full three-dimensional algorithm. The high recall values for the lower overlap regions of 0.5 and 0.75 are particularly important in the context of driver assistance systems, since a missed instance can lead to dramatic outcomes, as opposed to a not perfectly matched instance. The evaluation also demonstrated that the proposed *MCs* improved the results of the *FLIC* algorithm immensely and helped to detect otherwise missed objects.

The performance of both *FLIC* and the method by [45] drops noticeably in the 0.95 IoU bracket due to the underlying data. The *SemanticKITTI* dataset has a pre-applied ego-motion compensation,

Table 3.1.: **Comparison of the Segmentation Quality Using the Intersection over Union and the Recall Average.** The algorithms of [45] and [77] are directly compared to the *FLIC*

Method	$IoU_\mu \uparrow$ (No Ground)	$IoU_\mu \uparrow$ (Ground)	$R_\mu \uparrow$	$R_{0.5} \uparrow$	$R_{0.75} \uparrow$	$R_{0.95} \uparrow$
Bogoslavskyi et al.[45]	73.93	73.93	59.31	83.75	63.52	13.18
DBSCAN [77]	76.21	72.77	**76.50**	81.54	76.45	**69.25**
FLIC	76.20	72.31	63.73	84.30	67.51	22.03
FLIC (*1 MC*)	77.97	73.65	66.68	85.60	70.21	27.19
FLIC (*6 MC*)	81.14	75.48	71.92	88.25	74.99	36.05
FLIC (*14 MC*)	**84.25**	**76.39**	74.68	**89.75**	**77.61**	40.63

which slightly shifts and rotates the three-dimensional point cloud away from the original sensor configuration to compensate for movement. This manipulation of the point cloud hurts the performance of both methods in the 0.95 IoU bracket, which requires a precise projection of the raw data. The *DBSCAN* algorithm runs directly on the manipulated three-dimensional data and does not suffer from this issue.

3.4 Conclusion

This chapter presents a real-time algorithm for instance segmentation of lidar sensor data. The algorithm efficiently segments objects based on their three-dimensional distance using raw range images. A novel concept called "*Map Connections*" is introduced to make the approach more robust against over-segmentation. The method preserves three-dimensional information in two-dimensional representation for fast computation. The proposed approach outperforms comparable state-of-the-art methods in terms of speed, runtime stability, and instance segmentation performance. The algorithm has been utilized in various applications, including panoptic segmentation in Chapter 5.3, online detection systems for vulnerable road users in Chapter 8.2, improved non-causal object detection network [31], and test platform for collaborative perception algorithms [83].

4 Contributions to Semantic Segmentation:

Creating an Advanced Network Architecture for Three-Dimensional Semantic Segmentation of Lidar Point Clouds

In this chapter, a new network for semantic segmentation of lidar data is presented, which is designed to provide detailed and point-wise semantic information about the environment for robots and autonomous vehicles. The network is built with the aim of improving the current state-of-the-art in lidar semantic segmentation by leveraging the strengths of existing networks and combining their underlying principles in a synergistic way.

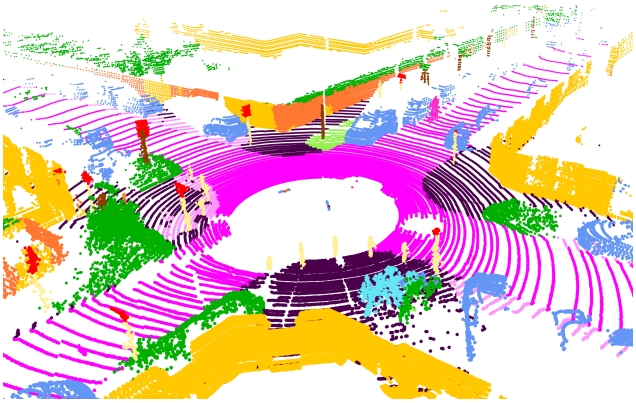

Figure 4.1.: **Semantic Segmentation Results of the Novel Neural Network Presented in this Chapter.** A three-dimensional point cloud is shown with semantic classified points. Every class is assigned a different color

To achieve this, a new module called *RangePillars* is introduced, which acts as a bridge between projection-, voxel-, and point-based methods for semantic segmentation. This module is a modification of *PointPillars* [127] and is based on a spherical coordinate system that allows for the combination of three-dimensional point cloud representations as voxels and range image-like grid structures.

The use of *RangePillars* enables the integration of various segmentation networks, such as point-based, range image-based, and voxel-based networks, in an end-to-end trainable fashion by combining the network data from all three configurations to leverage their respective strengths.

The following main contributions are provided in this chapter:

- A novel network architecture for semantic segmentation of lidar data.
- An evaluation of the proposed network architecture on open source lidar data.
- An extensive ablation study on the influence of each part of the proposed network architecture.

4.1 Related Work

Semantic segmentation of point clouds is the process of assigning a class label to each point in a point cloud, with the goal of dividing the point cloud into distinct regions or segments corresponding

to different objects or structures. There are three main categories of methods for performing semantic segmentation on point clouds:

Point-based Methods

Point-based methods operate directly on the individual points in the point cloud. *PointNet* [161] was the first deep learning architecture for this task. It did this by mapping all the points in the point cloud to a multi-feature space and using max pooling to extract a point cloud-wise feature vector. The extracted feature vectors are concatenated with the point-wise features and fed through a series of fully-connected layers which are used to classify the points. *PointNet++* [162] built upon this approach by using hierarchical farthest point sampling to create a pyramid structure similar to a *convolutional neural network* (*CNN*) which allows it to capture both global and local context in the point cloud. *KPConv* [15], on the other hand, uses *Kernel Point Convolutions* to operate directly with convolution operations on sampled point clouds spheres. These convolutions were applied to a given radius neighborhood of points, treating them as spherical operations.

Voxel-based Methods

Voxel-based methods for semantic segmentation of point clouds involve dividing the three-dimensional space occupied by the point cloud into a regular grid of small three-dimensional cells, or voxels [230]. The main advantage of voxel-based methods is that they can handle large, unstructured point clouds and can be efficient to process, as the voxel grid provides a fixed, regular structure that can be easily processed by standard machine learning techniques [127]. The progress of voxel-based processes was further accelerated by the advance of sub-manifold sparse convolutional networks [91] which allows convolutional operation even on very sparsely populated voxel grids, leading to very fine voxel sizes. *SPVNAS* [188] combines aggressive voxelization with a second point-based branch. The finer details captured by the point-based branch are added to the sparse voxel-based features to improve the final classification. *Cylinder3D* [228] partitions the point cloud into cylindrical partitions, taking advantage of the cylindrical point distribution often found in rotating lidar scanners. This partitioning allows for a more evenly distributed set of points, regardless of the range to the origin. Asymmetrical three-dimensional convolution was then applied to these cylindrical partitions, followed by a final point-wise refinement module to improve performance. *FusionNet* [222] fuses the point-wise inner-voxel aggregation features with the features of a multi-scaled voxel-based convolutional network to improve semantic segmentation. *(AF)*^{2}*-S3Net* [22] is a modified version of *MinkNet42* [10] with additional attention blocks that fuse the attention of different scales to improve the final classification.

Projection-based Methods

Projections-based methods use the two-dimensional panoramic range image projection of a point cloud. It provides a compact and efficient representation of the sparse and unevenly distributed point

cloud that is more efficient to process and analyze. One of the first approaches of semantic segmentation on range images was *SqueezeSeg* [211]. The *SqueezeSeg* architecture is a modified *SqueezeNet* [110] version with reduced parameter size for a lower computational complexity. *RangeNet++* [7] built upon the *Darknet53* [165] network architecture with adjustments to work with lidar projections images, and it added max pooling operations only along the horizontal dimension. *KPRNet* [120] combines a range image segmentation using a *ResNeXt-101* encoder [11] with a subsequent *KPConv* [15] layer before the final classification to reduce the errors caused by re-projections to the original point cloud. *SalsaNext* [61] replaces it's previous encoder-decoder structure from *ResNet* [18] blocks to residual dilated convolution stacks with gradually increasing receptive fields, the authors add a Baysian treatment to compute the uncertainty of the predictions via a point-wise *epistemic* and *aleatoric* uncertainty. *TORNADONet* [88] utilized a *PointPillars* [127] feature extraction on a cylindrical Bird-Eye-View projection of the original point cloud to replace the point features by voxel features for a following projection segmentation using a modified *SalsaNet* [3].

The work most similar to the approach presented in this chapter is *3d-MiniNet* [6], in which the authors project the three-dimensional Cartesian point cloud onto a range image and stack projected points in 3×3 grids to process them in a *PointPillars* [127] alike fashion to extract local three-dimensional features as additional input to the range image segmentation. Their concept is optimized for speed rather than segmentation quality, and they use the *PointPillars* solely as feature encoders. The *RangePillars* presented in this chapter on the other hand encode all points of the original point cloud, and can be used as feature encoders, but also as feature decoders, as which they improve the final segmentation quality.

4.2 RangePillars: Method

The *RangePillars* network is designed to use three-dimensional Cartesian point clouds as input data and provides point-wise semantic labels as direct output. It consists of three stages: (1) A pillar-voxel feature encoder network that converts a given three-dimensional point cloud to a list of voxels. (2) a two-dimensional convolutional backbone in which range image features are combined with the RangePillar-features; each pillar is used as a pixel and the high dimensional encoded features of the *RangePillars* represents the feature channels of every pillar. And (3) a point-wise classification head which combines the local point-wise pillar features with the point-wise projected pixel features for a final classification of each point.

4.2.1 Point Cloud to RangePillars

The *RangePillars* are designed as voxel representations for an evenly distributed grid structure of points in a three-dimensional voxel grid. To achieve this the Cartesian point cloud is projected

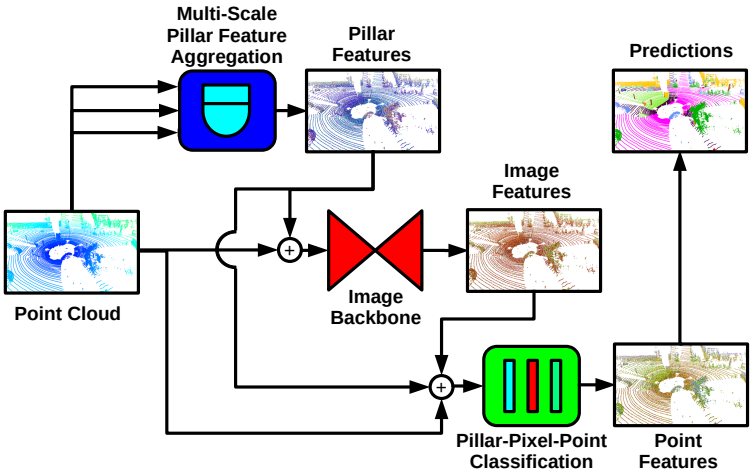

Figure 4.2.: Architecture of the *RangePillars* Network. The *Multi-Scale Pillar Feature Aggregation* module (blue) encodes the voxelized point cloud to extract pillar features. The image backbone (red) concatenates the range image projection of the original point cloud with the *RangePillars* features and processes it in an image encoder-decoder module. The *Pillar-Pixel-Point Classification* module (green) concatenates the pillar and image features with the point-wise information, to combine the information of all three modules for the final point-wise classifications. The plus sign denotes a concatenation along the feature dimension.

into the spherical coordinate system, by mapping every Cartesian point ($x, y, z \in \mathbb{R}$) to the spherical coordinates ($r \in \mathbb{R}^+; \phi, \theta \in \mathbb{R}$):

$$\begin{pmatrix} r \\ \phi \\ \theta \end{pmatrix} = \begin{pmatrix} \sqrt{x^2 + y^2 + z^2} \\ atan2(y, x) \\ asin(\frac{z}{r}) \end{pmatrix}, \text{ with } r \in [0, \text{inf}) \text{ and } \phi, \theta \in [-\pi, \pi], \tag{4.1}$$

where each point in the point cloud is transformed from the Cartesian coordinates (x, y, z) to spherical coordinates (r, ϕ, θ), where r represents the range, ϕ the azimuth angle, and θ the elevation angle. The azimuth and elevation angles are used to define bins in both directions, which are used to organize the spherical point cloud into a grid structure. This approach is often used as an intermediate step to turn a three-dimensional point cloud into a two-dimensional range image of the same [120, 215, 88, 3, 61, 6]. For the *SemanticKITTI* dataset, the scan-unfolding technique outlined in [19] was used to re-engineer the index of the lidar channels, as the dataset did not provide the original lidar channel indices in the data.

Figure 4.3 illustrates the distinction between Cartesian voxels, Spherical three-dimensional voxels, and the innovative *RangePillars* approach.

Usually, depending on the chosen bin size, each pixel in a range image is assigned the value of the last point of the bin or the point closest to the sensor. All other points of the bin are ignored and not represented in the range image.

The spherical grid structure on the other hand is simply a re-projection of the original point cloud. All points are taken into account for the creation of the *RangePillars*: the points are arranged along a new dimension, in which all points of a certain bin are grouped together. This allows a voxel

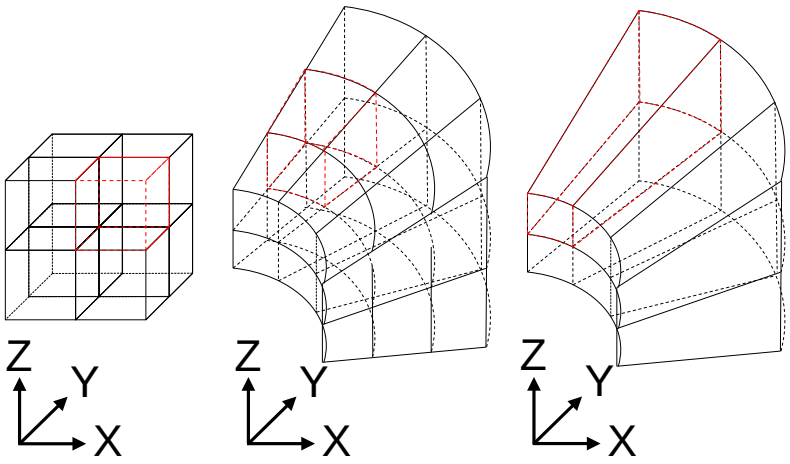

Figure 4.3.: **Comparison of Cartesian Voxels, Spherical three-dimensional Voxels, and *RangePillars*.** The figure showcases the differences between three voxel representation approaches: Cartesian Voxels, spherical three-dimensional Voxels, and the innovative RangePillars method. The diagram highlights the distinct structural characteristics and encoding schemes of each voxel representation. In each voxel grid, a single voxel is outlined in dashed red, providing a visual representation of the size of each voxel within the respective representation.

treatment to the spherical bins and assigns for each point the mean position in both the Cartesian x, y, z as well as in the spherical coordinates r, ϕ, θ and the pointwise offset to each of these mean values. The reflection intensity is not influenced by the spherical coordinate transformation and therefore added point-wise without a bin mean. This assigns the voxels with seven features per point.

Opposed to previous voxel methods [127, 6, 88, 188, 228] a generally uniform distribution of points is present in each bin, as the binning aligns to the recording method of the lidar sensor. A sub-sampling of voxels which have too many points or operations geared towards sparse matrices [91] are therefore not necessary. The *RangePillars* voxels keep all assigned points.

4.2.2 Multi-Scale Pillar Feature Aggregation

Depending on the chosen bin size in the voxelization of the point cloud, the resulting voxels can have multiple points per voxel, with a small total number of voxels or in the other extreme, only one point per voxel but a large amount of voxels. Both extremes are avoided by keeping 2 - 128 points in each voxel to avoid sub-sampling inside each voxel, while ensuring a minimum amount of points to enable a sufficient local point density for the inner-voxel operations.

To extract both fine-grained features from small-scale voxels with few points and rich contextual information from coarser voxel scales, the point cloud is voxelized using multiple *RangePillars* at different scales. These different *RangePillars* are then combined using a top-down, bottom-up approach similar to the one described in [124], with the scales of the pillars chosen to be multiples of each other. The largest scale *RangePillars* have the same resolution as a full-scale range image ($1S$: $64 \times 2048 = 131072$ voxels with 2 points each). Each *RangePillar* is processed via a mini-*PointNet*

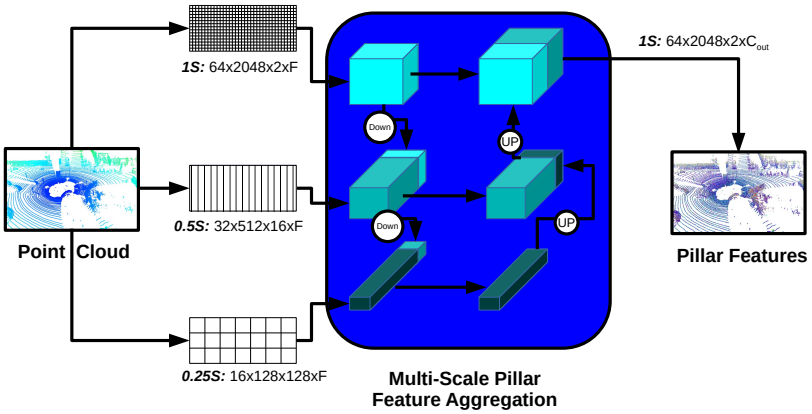

Figure 4.4.: Multi-Scale Pillar Feature Aggregation. The Multi-Scale Pillar Feature Aggregation module takes a point cloud that has been divided into three scales and encoded into range voxels as input. The highest resolution scale, called *1S*, is processed by a mini *PointNet* module and then downscaled to the *0.5S* scale using a two-dimensional convolution. The downscaled *1S* features are maxpooled along the point dimension and added onto the mid resolution scale, called *0.5S*, along the feature dimension. The resulting tensor is processed again by the same mini *PointNet* module as the higher scale voxels. The same process is repeated with the lowest scale, called *0.25S*. After the lowest scale mini *PointNet* module, the features are upscaled to the *0.5S* resolution and added onto the features of the *0.5S* mini *PointNet*, which is again processed by the same mini *PointNet*. This process is repeated for the *1S* scale pillar voxel features, which are the final output of the Multi-Scale Pillar Feature Aggregation module.

[161], and are down-sampled by a factor of 2 in the vertical and 4 in the horizontal direction to reach the mid scale *RangePillars* ($0.5S$: $32 \times 512 = 16384$ voxels with 16 points each). The two voxel scales are concatenated via their feature dimension, as described in [124]. This combines the local high-resolution voxel features with the mid-resolution features of the mid scale the same mini-*PointNet* of the higher resolution is applied to each pillar. This process is then repeated with the enriched mid scale *RangePillars* to reach the lowest scale pillars ($0.25S$: $16 \times 128 = 2048$ voxels with 128 points each).

To bring the context of the lower scale *RangePillars* back up to the higher scale pillars, a second, bottom-up path is applied. The coarse, lowest scale *RangePillars* are max pooled along their point dimension, up-sampled, and concatenated onto the mid scale pillar feature vectors. These *RangePillars* are again processed by a mini-*PointNet*. The process is repeated on these mid scale pillars and they are in turn up-sampled and concatenated onto the highest scale *RangePillars*, to add the context of the lower scale voxels as well as their own onto the fine scale pillars. The mini-*PointNet* are applied to this final highest resolution *RangePillars*. In this way the coarse pillars are enriched with the information of higher scale pillars, while the higher scale pillars gain the context information of the surrounding *RangePillars* of various resolution.

4.2.3 Image Backbone

The concept of the *RangePillars* is applied to the *SalsaNext* [61] range image semantic segmentation backbone: As a baseline a version of the *SalsaNext* has been adapted to work without the *RangePillars*, but with additional image-feature input layers: The spherical coordinate transformed point cloud as given in Equation 4.1 is mapped from $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ by

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{1}{2}[1-\phi\pi^{-1}]w \\ [1-(\theta+f_{up})f^{-1}]h \end{pmatrix}, \text{ with } u,v \in \mathbb{N}_+, \tag{4.2}$$

where (u,v) are the image pixel coordinates, (h,w) are the height and width of the resulting range image and $f = f_{up} + f_{down}$ is the vertical field-of-view of the lidar sensor. The image coordinates (u,v) are used to create a six channel image in the panoramic view. The input image consist of the inverted range and the intensity values similar to the recommendation of Kochanov et al. [120].

Two additional input channels are added to the input image, that encode the normal vectors of each point in the range image projection in relation to the lidar sensor. These are calculated by taking the mean of the angles spanned between the point in question and the two neighboring points:

$$N_{RP,hor}(u,v) = 0.5(A_{RP,hor}(u,v+1) + A_{RP,hor}(u,v-1)) \tag{4.3}$$

$$N_{RP,ver}(u,v) = 0.5(A_{RP,ver}(u,v+1) + A_{RP,ver}(u,v-1)), \tag{4.4}$$

where $A_{RP,hor}, A_{RP,ver} \in [-\pi,\pi]$ are the spanned angles between the point and its pixel space neighbors, and $N_{RP,hor}, N_{RP,ver} \in [-\pi,\pi]$ are the resulting relative normal vectors in relation to the sensor origin. They are split into the horizontal and vertical neighbor in the range projected image respectively. The angle images $A_{RP,hor}$ and $A_{RP,ver}$ are calculated via the law of tangents:

$$A_{RP,hor}(u,v) = \arctan\left(\frac{R_{PR}(u,v+1)\sin(\alpha_{hor})}{(R_{PR}(u,v) - R_{PR}(u,v+1)\cos(\alpha_{hor}))}\right) \tag{4.5}$$

$$A_{RP,ver}(u,v) = \arctan\left(\frac{R_{PR}(u+1,v)\sin(\alpha_{ver})}{(R_{PR}(u,v) - R_{PR}(u+1,v)\cos(\alpha_{ver}))}\right), \tag{4.6}$$

where R_{PR} is the range projected range image and $\alpha_{hor}, \alpha_{ver}$ are the horizontal and vertical lidar resolution respectively.

Further, two point-wise projected range distance images $D_{PR,hor}$ and $D_{PR\,ver} \in \mathbb{R}$ which encode the Euclidean distance of a point from it's horizontal and vertical range image neighbors are concatenated to the image features.

$$D_{PR,hor}(u,v) = \|(\mathbf{c}_{PR}(u,v) - \mathbf{c}_{PR}(u,v+1))\|_2 \tag{4.7}$$

$$D_{PR,ver}(u,v) = \|(\mathbf{c}_{PR}(u,v) - \mathbf{c}_{PR}(u+1,v))\|_2. \tag{4.8}$$

They are calculated by the L_2 norm of the projected Cartesian point coordinates $\mathbf{c_{PR}}$ and their neighboring pixel values in the horizontal and vertical respectively. The projected Cartesian point

coordinates $\mathbf{c_{PR}}$ are stored as a 3-channel image, in which each channel holds the information of the three coordinates x, y and z

$$c_{PR,hor}(0,u,v) = x \tag{4.9}$$

$$c_{PR,hor}(1,u,v) = y \tag{4.10}$$

$$c_{PR,hor}(2,u,v) = z. \tag{4.11}$$

This baseline image network with 6 input feature channels, was extended by adding the voxel-wise *RangePillar* features. The input image features are concatenated with the mean voxel features in the shape of pseudo-images resulting from the *Multi-Scale Pillar Feature Aggregation* step outlined in Section 4.2.2. The voxel-wise max features are assigned to the corresponding pixel values that are mapped from the voxels to the image pixels, as the number of the $1S$ scale voxels aligns with the number of pixels in the range projection image. Lastly, the classification head of the network is adjusted, to output the second to last layer as a feature-rich output for the point-wise classification network.

4.2.4 Pillar-Pixel-Point Classification

The point-wise classification combines the original point cloud with the output features of the *RangePillars* from the *Multi-Scale Pillar Feature Aggregation*, and with the pixel-wise output features from the adjusted image backbone. The *Pillar-Pixel-Point Classification* head uses the original point IDs and coordinates for each point in the output image as well as the *RangePillars*.

Given the original point cloud in the form $P \in \mathbb{R}^{N\times(3+C)}$, where N is the number of points, and C is the number of additional features to the three Cartesian coordinates, the coordinates of the range image projection and the *RangePillar* voxels can be projected into the same view. The coordinates of the range image projection are given in the form $RP \in \mathbb{N}_+^{N\times 2}$, where RP is in the same order as P. The *RangePillars* IDs are given in the form $RPV \in \mathbb{N}_+^{N\times V\times I}$, where V is the number of voxels and I is the number of inner-voxel points. These coordinates are used to re-project the features of each *RangePillar* to a point-wise list. The point-wise *RangePillar* features are concatenated with re-projected point-wise image output features and the points of the original point cloud.

The point-wise feature vector obtained by concatenating all three feature branches is processed by a one-dimensional convolution layer to output the probabilities for the final classification of each point. This point-wise segmentation combines the context-rich information from the image backbone with the three-dimensional information from the *RangePillars* and the original points, helping to avoid projection bleeding. This is a problem often seen in projection networks where borders between classes in the range image become smudged and one class bleeds into another, even when the points are far apart in the Cartesian point cloud [7].

4.2.5 Hydra Loss

Real world datasets are highly imbalanced which is challenging for deep neural networks. In order to reduce the bias to the more prominent classes, for the presented network, the strategy of *SalsaNext* was adopted, by which more value is added to the underrepresented classes by weighting the softmax cross-entropy loss L_{wce} as

$$L_{wce}(\hat{y}, y) = -\sum_{c=1}^{C} \alpha_c y_c \log(\hat{y_c}), \text{ with } \alpha_c = \frac{1}{\sqrt{f_c}}, \tag{4.12}$$

where C is the number of classes, y_c is the true label and $\hat{y}_c$ is the predicted label for class c, and α_c is the relative inverse square root of the class frequency f_c which is the share of points of the given class divided by the number of all points in the dataset.

Further, the *Lovász-Softmax loss* [8] L_{ls} is added to the *weighted categorical cross-entropy* L_{wce}.

The *Lovász-Softmax loss* is a method used in image segmentation to improve the accuracy of models. The key idea behind the *Lovász-Softmax loss* is to optimize the *Jaccard index*, also known as *Intersection over Union* (*IoU*). The IoU is a common metric for the quality of image segmentation, and it is more robust than simple pixel accuracy as it takes into account both the size and location of the predicted segments. This makes it a more suitable metric for imbalanced datasets or datasets with varying object sizes.

Given a vector of class probabilities $f(c)$ for each class c in C, and a ground truth label y, a vector of pixel errors $m(c)$ is constructed for each class c as follows:

$$m(c) = \begin{cases} 1 - f(c) & \text{if } c = y \\ f(c) & \text{otherwise.} \end{cases} \tag{4.13}$$

The *Lovász-Softmax loss* for class c is given by the Lov'asz extension of the Jaccard index for class c, applied to the vector of pixel errors $m(c)$:

$$L(f(c)) = \overline{\Delta J_c}(m(c)) \tag{4.14}$$

Finally, the overall *Lovász-Softmax loss* is the average of the class-specific losses:

$$L_{ls} = \frac{1}{|C|} \sum_{c \in C} \overline{\Delta J_c}(m(c)), \tag{4.15}$$

where $\overline{\Delta J_c}$ is the *Lovász extension* of the *Jaccard index* for class c, $f(c)$ is the vector of class probabilities for class c, $m(c)$ is the vector of pixel errors for class c, and C is the set of all classes. The sum is over all classes c in C, and $|C|$ is the number of classes.

The *Lovász extension* is a mathematical tool used to extend set functions to real-valued vectors. It is particularly useful for extending submodular functions, which are set functions that satisfy a natural

"diminishing returns" property. Submodular functions often arise when dealing with discrete objects like sets of pixels in an image or sets of points in a point cloud.

In simpler terms, the *Lovász extension* takes a vector of computed errors, orders them in decreasing order, and operates on these errors to compute a smooth approximation of the submodular *Jaccard Index*. The final loss is the average of the *Lovász-Softmax* losses computed for each class.

By directly optimizing the *IoU*, the *Lovász-Softmax los*s can lead to improved performance on image segmentation tasks, particularly when dealing with imbalanced or diverse datasets.

For a more detailed explanation of the *Lovász-Softmax loss*, the original paper [8] can be referred to.

Both losses, the *weighted categorical cross-entropy* and the *Lov'asz-Softmax loss*, are combined with a weighted influence for the final loss L

$$L = \lambda_{wce} L_{wce} + \lambda_{ls} L_{ls}, \tag{4.16}$$

where λ_{wce} and λ_{ls} scale the influence of the *weighted categorical cross-entropy* and *Lovász-Softmax loss* respectively.

The full network is trained in an end to end fashion which is causing considerable issues due to the internal projections and sub-networks: the weights of the *RangePillars* feature encoder are approaching zero values, due to the over-reliance of the full network on the image backbone predictions resulting from the range image projection. In order to force the network to use the *RangePillar* features and to lessen the bias towards the image backbone. The training loss is furthermore extended by a multi-headed loss which will further on be called *Hydra Loss*.

The *Hydra Loss* is the addition of loss terms on intermediate classification heads at the preliminary stages of the sub-networks. The original point cloud is encoded to the *RangePillar* voxels which uses the *Multi-Scale Pillar Feature Aggregation* to combine local features of the fine voxels with the global features of the coarser voxels grid. A classification head is added to the *RangePillar* feature encoder sub-network, in order to classify each voxel of the $1S$ voxels separately. The loss of the *RangePillar* Voxels L_V is combined by the two terms of the *weighted categorical cross-entropy* $L_{V_{wce}}$ and *Lovász-Softmax loss* $L_{V_{ls}}$, with a voxel weighting to the point-wise loss of the final full network.

The same process is repeated for the image backbone: the final feature output layer of the *SalsaNext* backbone is added to a separate image classification head, and both loss terms are calculated for this third head. The three heads are shown in Figure 4.5.

Each head's combination loss consists of the linear combination of a *weighted categorical cross-entropy* term and a *Lovász-Softmax loss* term:

$$L_V = \lambda_{wce} L_{V_{wce}} + \lambda_{ls} L_{V_{ls}} \tag{4.17}$$

$$L_I = \lambda_{wce} L_{I_{wce}} + \lambda_{ls} L_{I_{ls}} \tag{4.18}$$

$$L_P = \lambda_{wce} L_{P_{wce}} + \lambda_{ls} L_{P_{ls}}. \tag{4.19}$$

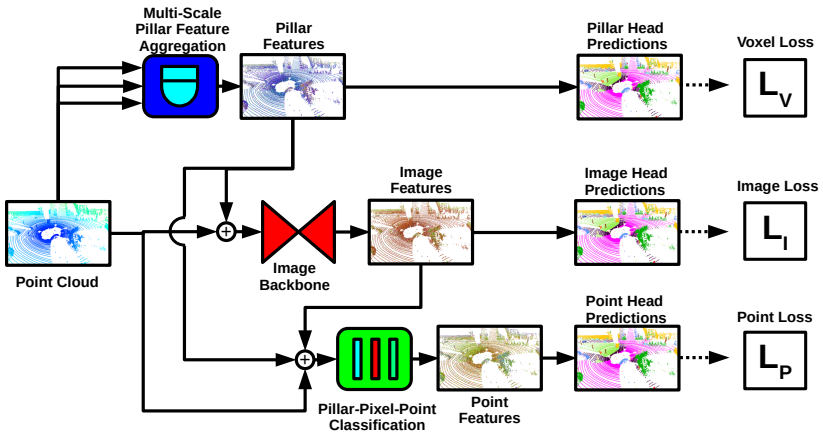

Figure 4.5.: Loss Calculations of Intermediate Network Heads. At training time, the intermediate features of the *RangePillars* and the image backbone are extracted and processed by classification heads. The resulting predictions are used to calculate a voxel loss L_V and an image loss L_I as well as the final point loss L_P.

The final multi-headed *Hydra Loss* L_H of the entire network therefore consist of the three loss terms, each adjusted by an origin loss weight

$$L_H = \lambda_V L_V + \lambda_I L_I + \lambda_P L_P \tag{4.20}$$

$$L_H = \lambda_V(\lambda_{wce} L_{V_{wce}} + \lambda_{ls} L_{V_{ls}}) + \lambda_I(\lambda_{wce} L_{I_{wce}} + \lambda_{ls} L_{I_{ls}}) + \lambda_P(\lambda_{wce} L_{P_{wce}} + \lambda_{ls} L_{P_{ls}}), \tag{4.21}$$

where the indices V, I and P denote the head-origin of each combination loss term and associated weight term of the voxel, image and point-wise head respectively.

4.3 RangePillars: Evaluation

The performance of the network and the ablation studies of its unique components were evaluated on the *SemanticKITTI* [39] dataset using the *mean Intersection over Union* metric (as described in Equation 2.31 on page 22). This metric measures the overlap between the predicted labels and the ground truth labels for each class and averages it over all classes. This metric is commonly used to evaluate the performance of semantic segmentation models.

To evaluate the influence of each component of the network and the various hyperparameters of the modules, ablation studies were conducted on a subset of the *SemanticKITTI* dataset. Only 10% of the original training data were used for these studies for a total of 100 epochs, as training multiple epochs for each configuration is computationally expensive and time consuming. The final trained *RangePillars* network which was evaluated on the challenge submission server in Section 4.3.3, was trained on the full training set. All networks were trained using a novel augmentation pipeline which is presented separately in Chapter 6.

4.3.1 Ablation Study

Ablation studies were conducted to understand the impact of *RangePillars* and their hyperparameters on network performance by modifying or removing network components.

The first experiment tested the *RangePillars* modules' impact on *mIoU*. Additional image features improved the *mIoU* from 46.8 to 47.3 by concatenating the proposed image input features to the input. The *RangePillars* were initially used as *Pillar Feature Encoders* before the image backbone, but did not improve the final *mIoU* (46.3) compared to the baseline network. Adding a set of *RangePillars* after the image backbone improved the *mIoU* from 46.8 to 57.8, demonstrating the positive impact of this approach on the network's performance.

The last variation of the *RangePillars* was using them on both sides of the image backbone as explained in Section 4.2. The loss parameters were $\lambda_{wce} = 1.0$ for the *weighted categorical cross-entropy* and $\lambda_{ls} = 2.0$ for the *Lov'asz-Softmax loss* in all three settings, these hyperparameter values were found to have the best influence on the performance and prior work [61] used the same weighting of the loss terms. The *Hydra Loss* for the third variation was set to reduce the influence of the voxel and image head from 1.0 to 0.0 dynamically after 10 epochs for the voxel head and 20 epochs for the image head to avoid the weights of these networks from vanishing. However, this setup did not improve the *mIoU* but instead reduced it to 37.4.

Based on these results, the *Pillar Feature Encoders* were dropped in further experiments, as they did not yield any improvements. The *Hydra Loss* was also replaced by only the point-wise loss combination, as it was found to be more effective.[1]

The second ablation study investigated the influence of the hyperparameters within the *RangePillars*. For these studies, the third row network in Table 4.1 was selected, in which the *RangePillars* were not used as an additional input feature pre-processing step for the image backbone but only as the final *Pillar-Pixel-Point Classification* head. The *mIoU* values for the various hyperparameter configurations are listed in Table 4.2.

By reducing the filter depth of the *RangePillars* from 64 to 32 the final *mIoU* score dropped by 2.1 points to 55.7. Increasing the depth from 64 to 128 on the other hand had no impact on the final *mIoU*, but the network converged quicker and reached the final performance on average 20 epochs earlier than the 64 filter depth version.

Increasing the number of layers inside the *RangePillars* did not yield an improvement in the final *mIoU*. The original configuration of a single layer performed better than the use of 2, 4 and 8 layers, that reached an *mIoU* of 56.6, 57.0 and 57,2 respectively. While the performance slightly increases with more layers, the original single layer *RangePillars* reached an *mIoU* of 57.8 with less computational demand.

The influence of the filter depth on the final *mIoU* appears to be very limited. Halving the number of filters leads to a worse *mIoU*, but doubling it has no noticeable effect. The number of layers after

[1] The last configuration was also tested with a sequential training approach, where the *Pillar Feature Encoders* were first trained with a Voxel head, then the image network was trained with the frozen *Pillar Feature Encoder* weights, and lastly, the second *RangePillar* layer was trained with point-wise classification. The full network achieved an *mIoU* of 56.5, but with three times the training iterations and thus not comparable to the other results in Table 4.1.

Table 4.1.: **Influence of the *RangePillar* Modules in the Full Network.** The usage of the *RangePillars* as voxel feature encoders has an almost not noticeable effect on the final segmentation quality, while the *Pillar-Pixel-Point Classification* usage of the *RangePillars* noticeably increases the final *mIoU*.

Position of the *RangePillars*	*mIoU*
Baseline *SalsaNext* [61]	46.8
Additional Image Inputs	47.3
Pillar Feature Encoder	46.3
Pillar-Pixel-Point Classification	**57.8**
Both	37.4

Table 4.2.: **Influence of Hyperparameters of the *RangePillars* on the Final Metric.** The *RangePillars* are used in the *Pillar-Pixel-Point Classification* step only and the influence of hyperparameter on the final *mIoU* is listed. Layer number one and filter depth 64, if not otherwise noted.

Hyperparameter	Value	*mIoU*
Filter Depth	32	55.7
	64	**57.8**
	128	**57.8**
Layer Number	1	**57.8**
	2	56.6
	4	57.0
	8	57.2

the *Multi-Scale Pillar Feature Aggregation* does not seem to have a clear influence on performance. Increasing the number of blocks to two, four, and even eight shows no clear trend and was discarded due to the increased computational effort required for training and inference. Therefore, the final *RangePillars* network had a layer number of one and a filter depth of 64.

4.3.2 Projection Cleaner

The ablation study of Table 4.1 has shown that the *RangePillars* are particularly useful for the *Pillar-Pixel-Point Classification* after the image backbone of the projection network. They present a good option to mitigate the main problem of projection networks: The projection bleed of the segmentation masks as shown in Figure 4.6. Two other modules have been published previously that try to solve this issue:

The first module is a depth dependent two-dimensional k-nearest neighbors [82] (kNN) post-processing module which was released with the network *RangeNet++* [7]. For each pixel in a range image, the classified label is compared with its direct pixel neighborhood and a depth-dependent weighting is applied to reduce projection bleeding.

The second module is based on *KPConv* [15] and was used in *KPRNet* [120] to improve classifications from a projection network re-projected onto the three-dimensional point cloud. *KPRNet* uses a three-dimensional kNN sampling to find the seven closest points for each point. These points are then processed in a *KPConv* layer to mitigate the bleed influence of the projection network.

The *Pillar-Pixel-Point Classification* module of this chapter falls into a similar category as the two mentioned modules. It was designed to combine the efficiency of a two-dimensional neighborhood search with the effectivity of a three-dimensional operation.

In Table 4.3 the raw projection model output as well as the three methods are listed with their influence on the final mIoU and the increase in runtime they cause. As can be seen, the implementation of

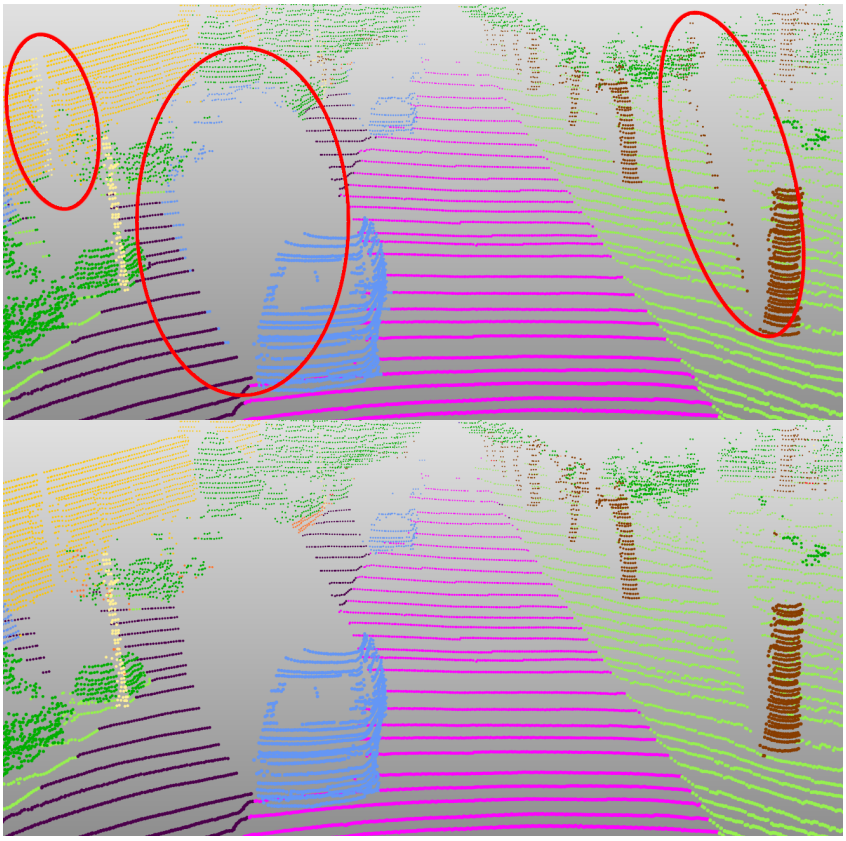

Figure 4.6.: **Segmentation Bleeding of a Projection Network.** Projection networks create segmentation masks similar to image segmentation networks. Points that are next to each other in the projection image can be very far in the three-dimensional point cloud (e.g., a car in front of a wall). A rough segmentation edge between two objects in the segmentation mask can therefore set a wrong classification to points that are very far away from the corresponding object (top). The *RangePillars* (bottom) act as a projection cleaning module, that reduces this effect.

Table 4.3.: **Comparison of the *Pillar-Pixel-Point Classification* Module with Current State-of-the-Art Post-Processing Modules for Projection Networks.** The usage of the *RangePillars* for the *Pillar-Pixel-Point Classification* yields the best enhancement of the final *mIoU* at a comparably low additional runtime. The additional CPU-bound runtime of the data-loader for the three-dimensional *k*NN sampling of *KPConv* as well as the *cython* [40] voxelization implementation of the *RangePillars* are added to the reported absolute runtime for a fair evaluation.

Post-Processing Method	*mIoU* ↑	$\Delta abs.\ mIoU$ ↑	$\Delta rel.\ mIoU$ ↑	*inference* [*ms*] ↓	*absolute* [*ms*] ↓
Base Network	46.8	+0.0	+0.0%	40	45
+ 2D *k*NN [7]	49.2	+2.4	+5.1%	**44**	**50**
+ *KPConv* Layer [15]	57.3	+10.5	+22.4%	69	102
+ *Pillar-Pixel-Point Classification*	**57.8**	**+11.0**	**+23.5%**	95	131

Table 4.4.: **The Performance of the Original *SalsaNext* and the Proposed *RangePillar* Network on the SemanticKITTI Validation Data.** Both the proposed *RangePillar* as well as the provided checkpoint of the *SalsaNext* were inferred on the validation set of the dataset.

Methods	*mIoU* ↑	car	bicycle	motorcycle	truck	other-vehicle	person	bicyclist	motorcyclist	road	parking	sidewalk	other-ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign
SalsaNext [61]	56.9	86.7	**40.7**	42.0	79.3	42.5	**64.6**	69.4	0.0	94.5	42.6	80.2	3.5	80.6	48.3	80.8	**61.6**	65.1	53.1	**44.9**
RangePillars	**61.3**	**94.0**	31.4	**57.3**	**86.6**	**58.3**	62.5	**75.5**	**0.6**	**94.9**	**51.5**	**82.2**	**5.9**	**88.9**	**54.8**	**85.7**	61.3	**73.1**	**55.4**	44.5

the *RangePillars* with an *mIoU* of 57.8 outperforms both the two-dimensional *k*NN post-processing module (49.2) as well as the *KPConv* projection cleaner network (57.3).

The inference time increase of the two-dimensional *k*NN post-processing module is neglectable with 5 ms, while both the *KPConv* projection cleaner network as well as the presented *Pillar-Pixel-Point Classification* increase the runtime beyond a real-time capable application with an additional 57 and 86 ms, respectively. Both three-dimensional modules are therefor preferable for offline applications, while the two-dimensional *k*NN enables an online projection cleaning at a lower performance.

4.3.3 Benchmark Results

Based on the results of the ablation studies on the validation set in Section 4.3.1, the best combination of modules and hyperparameters was chosen to be retrained on the full training set of the dataset. The resulting network achieved an *mIoU* of 61.3 on the validation set, 4.4 point higher than the original *SalsaNext* with the two-dimensional *k*NN projection cleaning module with an *mIoU* of 56.9.

The proposed additional *RangePillar* network on top of the original *SalsaNext* improves most classes, as it prevents the projection bleeding effect much better than the original two-dimensional *k*NN projection module of the *SalsaNext*, as shown in Table 4.3.

4.4 Conclusion

The *RangePillars* network is not the new State of the Art for semantic segmentation. The network was designed to combine a synergy of benefits from different network architectures. The evaluations show that the combination of the range image and *RangePillars* features in the encoder of the image backbone do not provide additional benefit, but on the contrary appear to degrade the performance of the image backbone by introducing more noise than additional useful features.

The use of the proposed *RangePillars* as a post-processing module in a projection based network via a *Pillar-Pixel-Point Classification* improved the final classification to a higher degree than comparable modules.

Several new approaches have been tried in the development of this network, but in the meantime the State of the Art has advanced as well. The advantages of the different network types mentioned in the introduction have been recognized and combined in several new networks. *3d-MiniNet* [6] achieved an improvement in runtime by replacing the early sections of the encoder of a range image network with a pseudo voxel architecture that uses pixel values within a range image as points. By doing this, the authors avoid the overhead of voxelization and tensor re-structuring which caused major problems in the development of the *RangePillars* network. *RPVNet* [16] has achieved a combination of point-wise, image-wise and voxel-wise sub-networks, by multiple fusions of the features in all three strands in different locations within the network. It is clear to see that the base idea that inspired the development of the *RangePillars* was also recognized as a desirable synergy by others.

In conclusion the *RangePillars* are a promising additional network module that enhances projection based networks by combining projection image features with point-wise and voxel-wise features for precise point-wise semantic segmentations. Furthermore in the development of this network, the foundation for further methods were developed which led to new augmentation methods for the training of neural networks for lidar semantic segmentation which are presented separately in Chapter 6.

5 Contributions to Panoptic Segmentation:

Developing Novel and Improved Methods for Panoptic Point Cloud Segmentation

Panoptic lidar segmentation combines semantic and instance segmentation by assigning each point a semantic class and instance label. This process enables machines to understand their three-dimensional surroundings using lidar data, making it a critical component of computer vision. Recent years have seen a surge in the development of deep learning-based networks, greatly improving the accuracy and efficiency of panoptic lidar segmentation. In this chapter, the latest advances in this area are explored and new network combinations for panoptic lidar segmentation are designed and implemented.

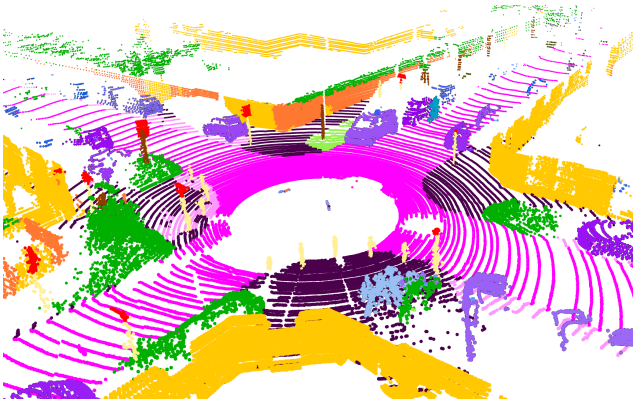

Figure 5.1.: **Panoptic Segmentation Results of One of the Novel Method Combinations Outlined in this Chapter.** A three-dimensional point cloud is shown with semantic classified and instance-wise separated points. Every instance of the same class is assigned a slightly different shade of the same color.

Section 5.1 provides an overview of the current State of the Art of panoptic lidar segmentation and outlines the three main methods for panoptic lidar segmentation.

In Section 5.2, a state-of-the-art heuristic lidar clustering algorithm is used to extract object clusters, which are subsequently classified using a CNN image classification network. This approach results in an efficient panoptic lidar segmentation that can run on a single CPU core in real-time.

In Section 5.3, two state-of-the-art semantic segmentation networks are combined with heuristic lidar clustering algorithms, and a parallel GPU approach is developed for the lidar clustering algorithm discussed in Chapter 3.

The following main contributions are provided in this chapter:

- A novel CPU-based system for real-time panoptic segmentation of lidar data.
- A novel GPU-based Euclidean clustering algorithm inspired by the *FLIC* algorithm.
- A novel GPU-based two-step combination system for panoptic segmentation.
- Evaluations of both proposed system architectures on open source panoptic lidar data.
- A detailed comparison with state-of-the-art panoptic lidar segmentation methods.

5.1 Related Work

There are three main approaches to panoptic segmentation: sequential, two-stage, and single-stage.

Sequential approaches handle the semantic segmentation independently of the instance segmentation by combining a semantic segmentation with the object detection results of bounding box detection algorithms. Examples of such a sequential combination were presented in [147], in which the authors combined the *RangeNet++* [7] and *KPConv* [15] semantic segmentation networks with the *PointPillars* [127] object detection network.

Two-stage approaches, such as *EfficientLPS* [181] and *MOPT* [109], detect foreground instances, refine them, and then apply a semantic segmentation to the remaining background points.

Single-stage approaches, on the other hand, use unified networks to detect both the semantic and instance segmentation in a single backbone. They usually use two heads for the final task split. Examples of single-stage approaches include *Panoptic RangeNet* [147], *Panoster* [85], *DS-Net* [105] and *Panoptic-PolarNet* [232].

There is also a special type of network for panoptic segmentation that combine heuristic clustering algorithms with semantic segmentation and instance refinement modules. Prominent examples of this type of network include *Panoptic4D* [36], *GP-S3Net* [164] and *Panoptic-PHNet* [132].

5.2 Lidar Cluster Classification

The lidar cluster classification [93] was the first reported method for panoptic segmentation, that was able to run on a single CPU core in real time. The work outlined in this section was already published in a conference paper [93] as well as in the doctoral thesis [92] of the first author of the same. Several of the figures and tables in this section were reproduced from the referenced conference paper.

In this section a real time capable panoptic lidar segmentation algorithm is outlined. The method combines the clustering algorithm described in Chapter 3 with a second stage that extracts important features for a computationally efficient classification of the segmented lidar clusters. An example of a panoptic segmented point cloud of the *SemanticKITTI* [39] dataset with this method can be seen in Figure 5.2, compared to the same scene with the ground truth labels.

5.2.1 Lidar Cluster Classification: Method

The method consists of two parts. The first part, the class-independent clustering of objects, has already been presented in Chapter 3. The clustering step yields a point cloud with a ground segmentation and individual cluster labels or each object that is not part of the ground.

The second part, the classification, builds on the instance-wise segmented point cloud. More precisely, on the range image projection of the point cloud. The classification step is performed on the range

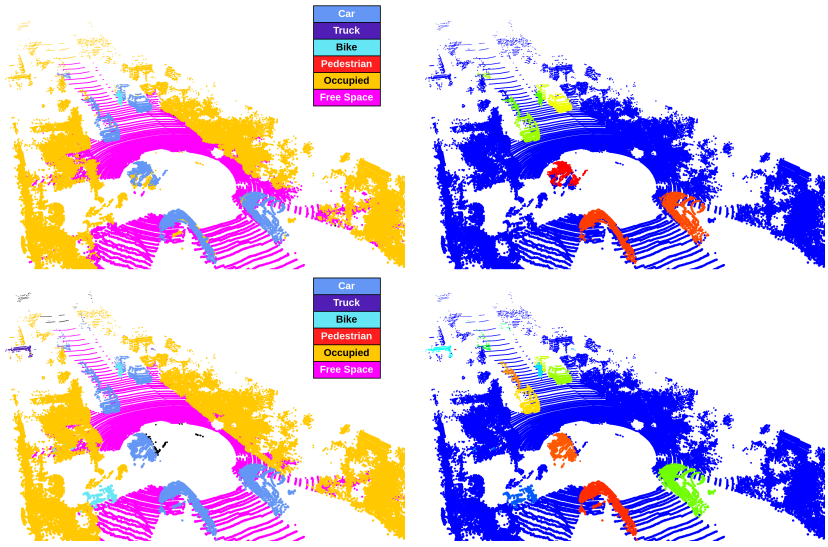

Figure 5.2.: **Panoptic Segmentation of the Lidar Cluster Classification Method Outlined in Section 5.2.1.** The semantic class mapping was adjusted as shown in Figure 5.5. The predictions of the methods are shown in the top row with the semantic (left) and instance predictions (right) shown separately. The remapped ground truth is shown in the bottom row. Please note that the instance IDs are randomly colored.

image, similar to the clustering step. This reduction of the three-dimensional point cloud to the two-dimensional representation is one of the main reasons for the high efficiency of the method.

The second reason for the efficiency of the proposed method is the re-formulation of a segmentation problem to a classification problem. The preceding clustering algorithm outputs instance-wise pixel segments within a range image. These masks are used to crop out the corresponding objects from the range image to stand-alone images which are processed with a standard classification *CNN*.

Instead of the three color channels consisting of a red, a green and a blue channel, a combination of three different feature layers is used: the lidar remission and two values representing the surface normal structure. The remission is available in range coordinates by projecting the raw point-wise remission values of the lidar point cloud. The surface structure is given in the form of the relative normal vector scalar components of each point in relation to the sensor.

These are calculated by taking the mean of the angles spanned between the point in question and the two neighboring points:

$$N_{RP,hor}(u,v) = 0.5(A_{RP,hor}(u,v+1) + A_{RP,hor}(u,v-1)) \tag{5.1}$$

$$N_{RP,ver}(u,v) = 0.5(A_{RP,ver}(u,v+1) + A_{RP,ver}(u,v-1)), \tag{5.2}$$

where $A_{RP,hor}, A_{RP,ver} \in [-\pi,\pi]$ are the spanned angles between the point and its pixel space neighbors, and $N_{RP,hor}, N_{RP,ver} \in [-\pi,\pi]$ are the resulting relative normal vector scalar components in relation

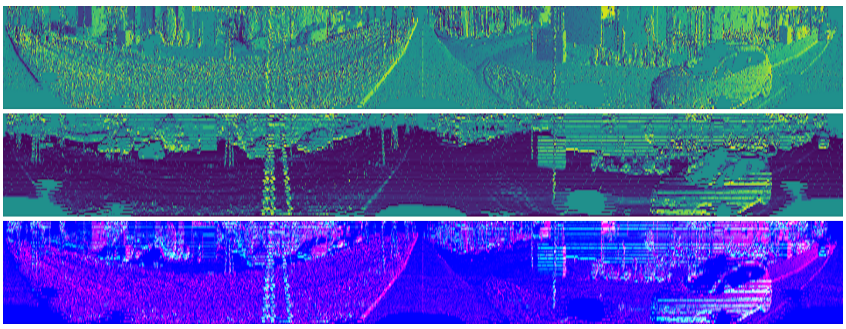

Figure 5.3.: **Visualization of Normal Vectors.** The horizontal *(top)* and vertical *(middle)* normal vector component image, as well as a combined view of both components *(bottom)* are visualized.

to the sensor origin. They are split into the horizontal and vertical neighbor in the range projected image respectively. The angle images $A_{RP,hor}$ and $A_{RP,ver}$ are calculated via the law of tangents:

$$A_{RP,hor}(u,v) = \arctan\left(\frac{R_{PR}(u,v+1)\sin(\alpha_{hor})}{(R_{PR}(u,v) - R_{PR}(u,v+1)\cos(\alpha_{hor}))}\right) \tag{5.3}$$

$$A_{RP,ver}(u,v) = \arctan\left(\frac{R_{PR}(u+1,v)\sin(\alpha_{ver})}{(R_{PR}(u,v) - R_{PR}(u+1,v)\cos(\alpha_{ver}))}\right), \tag{5.4}$$

where R_{PR} is the range image and $\alpha_{hor}, \alpha_{ver}$ are the horizontal and vertical lidar resolution respectively.

An exemplary representation of both normal component images $N_{RP,hor}$ and $N_{RP,ver}$ can be seen in Figure 5.3, with an additional combined view for illustration purposes.

These simple calculations and the projection of the raw remission data results in a range image projection image with three data channels: remission, vertical normal vector and horizontal normal vector. Furthermore, a binary mask representing the shape of the instances is used, but not added to the classification image. Based on the binary instance mask, the image is cropped to the instance, and multiplied with the binary instance mask to remove the background in all three layers of the cropped images. These are then classified by a small *CNN*.

The main objective of this work is to facilitate a fast classification of lidar instances which is capable of running on CPU in real time. In [93] a *mini-XCEPTION* [34] variation with fewer separable modules was used together with a second network branch that used statistical cluster information such as instance size and point density. In this work the network does not use the second branch and relies entirely on the cluster instance classification via the *CNN*. The architecture is shown in Figure 5.4. The reduced *mini-XCEPTION* [34] consists of two residual separable modules with a maximum filter depth of 64. The complete *CNN* architecture has a total of 20,802 parameters. By using depth-wise separable convolutions, in which each feature dimension is convoluted separately in two-dimensional operations, the computation demand reduces immensely compared to normal convolutions across all channels [56]. The *CNN* classifies each cluster instance image to one of five classes: "Car", "Truck", "Pedestrian", "Bike" or the "None" class which denotes an object that is not one of the four defined ones.

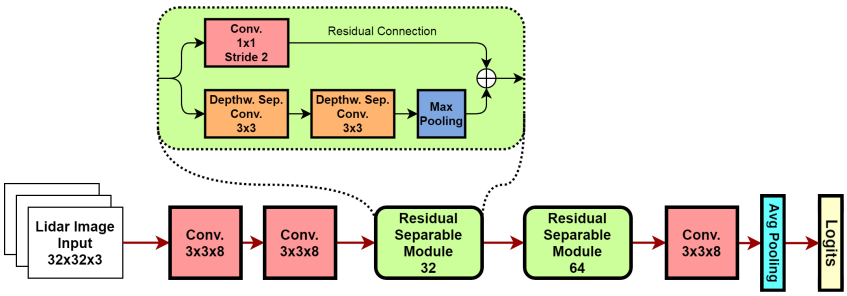

Figure 5.4.: **Network Architecture.** The *CNN* is a lightweight version of a *mini-XCEPTION* [34] variation.

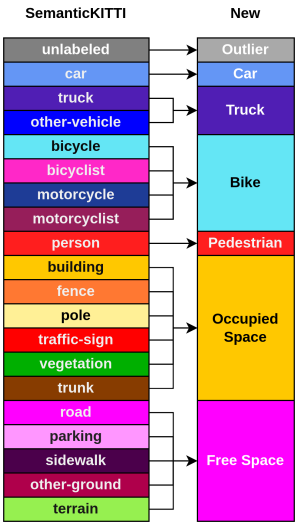

Figure 5.5.: **Reduced Class Mapping of the *SemanticKITTI* Dataset.** The classes used in the training and evaluation were remapped to match a subset of dynamic road user classes as well as two background classes indicating free and occupied space respectively.

5.2.2 Lidar Cluster Classification: Evaluation

The described method was evaluated using the *SemanticKITTI* [39] dataset. The semantic classes of all points were mapped to the five classes of the classification network as well as a general ground class as shown in Figure 5.5. The classification network has been trained with the annotated point clouds of the available training logs "00" to "10" with the exception of log "08" which was kept for validation.

Table 5.1 shows the results for the class-wise semantic segmentation *Intersection over Union* (*IoU*) metric of the combined approach of clustering the point cloud and classifying each cluster separately on the validation log of the dataset. The semantic segmentation metric was chosen for the validation, as only the class labels are trained by the *CNN*, while the instances are extracted by the heuristic clustering approach. The ground *IoU* is not listed, as it results from the ground extraction of the clustering method described in Chapter 3.2.1.

Table 5.1.: **Semantic Segmentation Results.** The average class-wise *Intersection over Union* (*IoU*) on the re-mapped validation set. The results are reported once for the clustered instances of [99] and once for the ground truth instances as input for the classification approach of this section.

Method	None	Car	Truck	Bike	Pedestrian
FLIC Instances [99]	95.4	75.0	47.2	26.5	28.2
GT Instances	99.4	92.6	73.2	52.5	55.8

Table 5.2.: **Panoptic Segmentation Results.** The results on the *SemanticKITTI* test set are shown for the four metrics *panoptic quality (PQ), segmentation quality (SQ), recognition quality (RQ)* and *mIoU*.

Class	PQ ↑	SQ ↑	RQ ↑	mIoU ↑
Car	75.4	86.6	87.0	79.2
Truck	5.3	88.8	6.0	3.7
Bike	8.2	72.3	11.4	4.6
Pedestrian	37.7	90.5	41.7	16.1

The score of this metric was computed for two approaches. First the full method, in which the instances were clustered with the *FLIC* clustering algorithm of [99] as described in Chapter 3 and classified with the *CNN* of Section 5.2.1. The second approach applied the classification directly to the annotated ground truth instances in the dataset. This allowed for a comparison, on how the performance was influenced by the quality of the provided object instances. As the results in Table 5.1 show, the performance of the classification depends directly on the segmentation quality of the given object clusters. Especially the "Bike" and "Pedestrian" class show significant reliance on the instance segmentation quality as the *IoU* drops from 52.5 to 26.5 and from 55.8 to 28.2 respectively. The other classes "Car", "Truck" and "None" also drop by 17.6, 26.0 and 4.0 *mIoU*.

As the resulting class labels are provided instance-wise, the results can be directly used as panoptic segmentation. The full method was validated on the panoptic segmentation benchmark of the *SemanticKITTI* dataset. This challenge uses the *Panoptic Quality* (*PQ*) as described in Equation 2.32 on page 23.

The presented approach was limited to the class set shown in Figure 5.5. The results in Table 5.2 were therefore reverse mapped to 4 of the 19 original classes of the *SemanticKITTI* dataset: "Car", "Bicycle", "Truck" and "Pedestrian". All other classes were mapped to "Unknown".

The reverse mapping introduced wrong predictions to the "Truck" and "Bicycle" classes, as the network merges the two original classes "other-vehicle" and "truck" to "Truck" as well the four classes "bicycle", "bicyclist", "motorcycle" and "motorcyclist" to "Bike". Therefor three out of four correct but re-mapped "Bike" predictions of the network are "wrong" for the evaluation server which hurt the performance of these two classes.

The classification network presented in Section 5.2.1 was implemented in *Python* with *Tensorflow* [1] and *Keras* [@57], devoid of any customization or optimizations. Most of the point clouds in the dataset yielded fewer than 100 clusters. This is representative for automotive lidar scenes of inner cities, suburbs and highways. The inference of 100 randomly selected input instances on the proposed lightweight version of a *mini-XCEPTION* was measured multiple times on an *Intel*

i7-6820HQ laptop CPU @ 2.70 GHz and an average inference time of 32 ms was recorded. The presented combination of the *FLIC* algorithm presented in Chapter 3 and the classification network described in Section 5.2.1 is able to create a panoptic lidar segmentation in real-time entirely on CPU.

5.3 Lidar Image Panoptic Segmentation

Panoptic segmentation is a combination of semantic segmentation and instance segmentation. This task is usually performed in a single network as described with multiple examples in Section 5.1, but a multi-step process is equally valid to reach the goal. In the previous Section 5.2, a multi-step process was presented which was designed for fast inference without the use of a GPU.

In this section a combination of state-of-the-art networks for semantic segmentation and a subsequent instance segmentation is outlined. The combination of both tasks yields a panoptic segmentation of lidar data with minimal additional overhead.

Furthermore a revised version of the algorithm proposed in Chapter 3 is described in Section 5.3.1. In Section 5.3.2 the combination method of this Section is evaluated with two different networks for semantic segmentation each combined with two different instance segmentation clustering methods. This is done with respect to their final panoptic segmentation quality as well as the runtime of the entire combination method. All four combinations are compared to current state-of-the-art panoptic segmentation methods. The proposed combination method is one of the top three published methods of the *SemanticKITTI* challenge with a *Panoptic Quality* (*PQ*) of 58.0, while another combination of the proposed approach is currently the fastest method with a frequency of 37.8 lidar frames per second at a respectable Panoptic Quality of 45.0.

5.3.1 Lidar Image Panoptic Segmentation: Method

For the combination methods presented here, two publicly available networks for semantic segmentation of lidar data [228, 133] are combined with two class-independent instance clustering methods. The two semantic segmentation networks are chosen because one has a very high semantic segmentation quality [228] and the other a very high inference speed [133]. Both were made open-source by their authors[1] [2].

Due to the fact that both selected semantic segmentation networks [228, 133] are running on a GPU, the panoptic segmentation task and therefore also the instance segmentation is not restricted to CPU based hardware as opposed to the method of Section 5.2. The instance clustering method of Chapter 3 is therefore revisited and reformulated in Section 5.3.1. First, however, the two open-source lidar semantic segmentation models used will be presented.

[1] https://github.com/xinge008/Cylinder3D
[2] https://github.com/sj-li/MINet

Cylinder3D

Cylinder3D [228] is a voxel-based semantic segmentation network that utilizes a three-dimensional version of the cylindrical partitioning and processing of point clouds as proposed in *PolarNet* [225]. By dividing the point cloud into cylindrical voxels and extracting features from the resulting polar grid, the network is able to effectively encode and classify the data. The feature extraction process utilizes a *PointNet*-alike [161] multi-layer perceptron to process each voxel, and the resulting features are fed into an encoder-decoder structure that is similar to a convolutional neural network used in image processing. However, unlike traditional image *CNNs* which operate on a two-dimensional grid, *Cylinder3D* operates on a three-dimensional voxel grid in polar coordinates. The final classifications are refined for each individual point in the publication, but in the open-source implementation[3] used in this chapter, the classifications are output voxel-wise instead. In order to improve performance, the network was retrained using an augmentation pipeline described in Chapter 6.

MINet

MINet [133] is an architecture for the semantic segmentation of projected lidar point clouds in a range image structure. It uses a multi-scale approach, in which a special focus was put on the computational operations to be as efficient as possible and to avoid any redundant computation. The authors created additional supervision of the intermediate layers, for better convergence of the weights without additional runtime of the inference. The *MINet* architecture is able to process lidar data many times faster than the recording time of the sensor at a comparably high *mIoU*. In order to improve performance, this network was also retrained using the same augmentation pipeline as above which is outlined in Chapter 6.

FLIC++

The basic principle of the clustering method of Chapter 3 was revised. Instead of a sequential iterative reformulation of the clustering to a *connected-component labeling* problem as done in Chapter 3.2.2, the connectivity definition of the individual points is approached as a parallelizable method in this section.

The basic assumption remains the same, namely that the lidar point cloud is still projected in the form of a range image for a structured and lower-dimensional representation. Likewise, the Euclidean distance between the lidar points is used as a decisive criterion to decide whether two points originate from the same object. However, unlike in Chapter 3.2.2, the metric is not further reduced to the squared distance as a function of the lidar sensor parameters and depth values as originally shown in Equation 3.5 on page 34. Rather, the three-dimensional Euclidean position values of all lidar points are used in the form of a multichannel image corresponding to the structure of a range image. This means the projected image does not have one value per pixel, but three separate values for the x y and z Cartesian coordinates in a structure similar to an RGB image. This enables the calculation

[3] https://github.com/xinge008/Cylinder3D

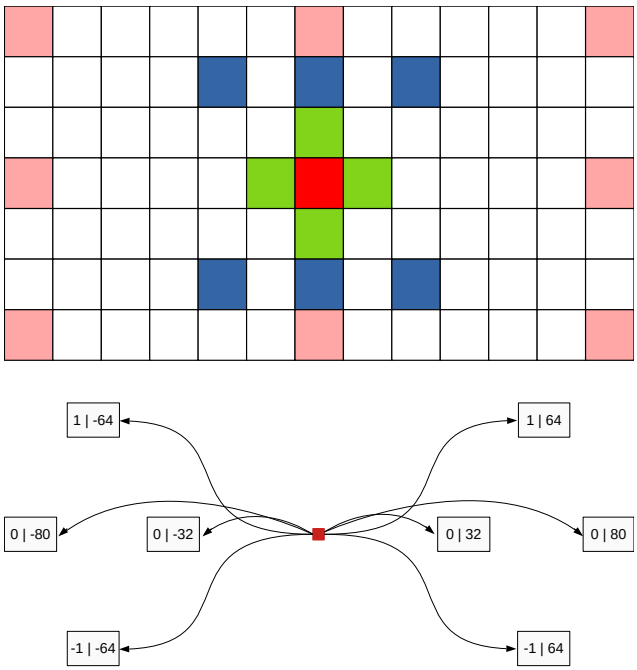

Figure 5.6.: ***FLIC++* Pattern of the *Map Connections*.** The concept of the *Map Connections* of Chapter 3.2.2 was extended for more connections beyond direct neighbors in the near vicinity (top). The *Scout Connections* (bottom) extend to very far points relative to each lidar point in order to bridge larger gaps resulting from occlusion or missing return values. The numbers in the bottom image correspond to the relative pixel offset to the *Scout Connections* in the vertical and horizontal pixel space.

of the Euclidean distance d between two lidar points $\mathbf{p}$ and $\mathbf{q}$ directly via the absolute difference of their coordinates:

$$d(\mathbf{p},\mathbf{q}) = |\mathbf{p}-\mathbf{q}| = \sqrt{(\mathbf{p}-\mathbf{q})^2}, \tag{5.5}$$

where $\mathbf{p},\mathbf{q} \in \mathbb{R}$ are the three-dimensional position vectors of each point in the range-projected image of the Cartesian coordinates.

One of the key findings in the evaluation of the *FLIC* algorithm in Chapter 3.3 was the improved segmentation quality by adding *Map Connections* (*MCs*). Table 3.1 on page 43 showed that with more *Map Connections* between distant lidar pixels in the range image, the segmentation quality increases due to a reduction of the over-segmentation of contiguous objects that do not have direct connections in the range image.

In the present new revision of the clustering method, the main focus is therefore on enabling more *MCs* beyond the immediate neighbors. The new approach does not subsample the depth image to a sub-connection of a few individual *MCs*, but allows each individual lidar point to connect to its direct neighbors and all defined *MCs*.

For this purpose, the connection pattern shown in Figure 5.6 is selected, as it considers both multiple dense connections of the immediate surroundings (top), but can also bridge large occlusions by other objects with more *MCs*. The latter is extended in this revision of the algorithm by very remote *Map*

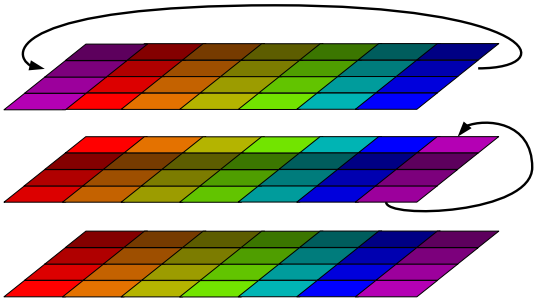

Figure 5.7.: **Schematic Visualization of the Image Wrap Around.** The *Map Connections* and *Scout Connections* are brought to the target pixel position by copying the original image and shifting the rows and columns according to the *Connection* position.

Connections which are further on also called *Scout Connections* (*SCs*). These add information of "far" pixels in the projection image representation to bridge gaps created by occluded parts of an object.

In order to compare an origin pixel with each of its connections, the Euclidean position image is copied for each connection, and the rows and columns of the copies are shifted, and "*wrapped around*" accordingly. This is done by shifting the copies of the original three-channel Euclidean image containing the point coordinates as shown in Figure 5.7 for a vertical and a horizontal connection. The pixel position (u, v) in the original Euclidean image corresponds to the pixel positions of the connection pixels in the shifted copy images.

These shifted images are stacked with the original image along a new, fourth dimension. Equation 5.5 is applied to this densely populated tensor along the new dimension. This reduces the four dimensional tensor again to a three-dimensional image which contains the distance to each *Map* and *Scout Connection* along the channel dimension of the image. The ordering of the channels corresponds to the original ordering of the stacked shifted images.

With a defined threshold value the three-dimensional distance-encoding tensor is converted to a binary one. Therefore as shown in Figure 5.8 a vector is created for each pixel position in the original projection image, that corresponds to the binary decision of connection or separation from the given pixel position to it's connected pixels.

The same process is repeated with a second image that assigns ascending integer number values to each pixel which is called *ID image* in the following. By shifting the *ID image* copies and stacking them to an *ID tensor*, the channel positions of the IDs correspond to the same channel positions in the binary connection tensor. This commonality allows to multiply the binary tensor with the *ID tensor* and to thus zero all *ID values* that have no connection to the target pixel as shown in Figure 5.8.

Lastly, the ID tensor is reduced to its maximum value along the stacked ID channel dimension. This assigns to each pixel position of the original projection image the maximum identity value of all its connections. By repeating this process of assigning the IDs and reducing them to their maximum along the stacked dimension several times, high ID values propagate along all connections until all

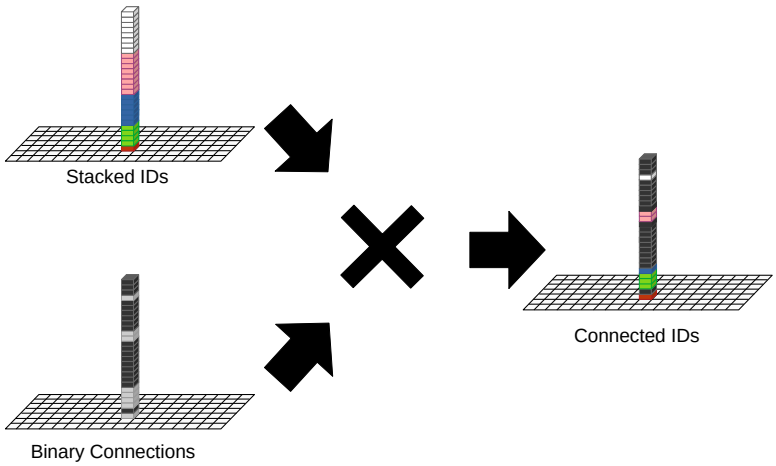

Figure 5.8.: **Stacked Pixel ID Values of *FLIC++* Connections**. The IDs are taken from the shifted connection positions as shown in Figure 5.6, stacked along a new axis and multiplied by the binary value of each connection. This results in a vector of every pixel ID that is connected to the target pixel via the chosen clustering metric, and zeros for those that are not connected.

points of a cluster have the corresponding largest ID value, as the lower values are displaced by the maximum operation.

This novel process replaces the problem formulation of a *connected components labeling problem* to an entirely new approach. The whole process is implemented with parallelizable operators in the deep learning framework *PyTorch* [155]. The new algorithm is called *FLIC++*, as the underlying metric is the same as *FLIC*, but the novel approach enables a fast GPU based clustering with more *Map Connections* and *Scout Connections* which results in a noticeably higher segmentation quality as shown in the evaluation in Section 5.3.2.

Combination

The last step of the proposed method for panoptic segmentation is the combination of the mentioned semantic segmentation networks with either the presented novel *FLIC++* or the original CPU-bound *FLIC* of Chapter 3. One of the main weaknesses of the *FLIC* algorithm was the influence of the ground extraction on the object separation quality as can be seen in Table 3.1 on page 43. The ground segmentation is therefore removed in the further development of the process. Instead, the semantic segmentation output of the preceding network is used to mask out any point of the ground classes from the point cloud presented to the instance segmentation part.

The combination method is therefore a sequential process using semantic segmentation to hide both ground points and points of classes for which instance separation is not required. The lidar points that are part of the classes that can be instantiated are passed to the instance separation as a projection image. This reduces the instance separation process to the bare minimum, since only a fraction of the original lidar points are used in the form of a depth image for the *FLIC* algorithm or in the form of an Euclidean position image for the *FLIC++*. The instance separations is applied and finally combined with the entire semantic labels of the previous network for a panoptic segmentation.

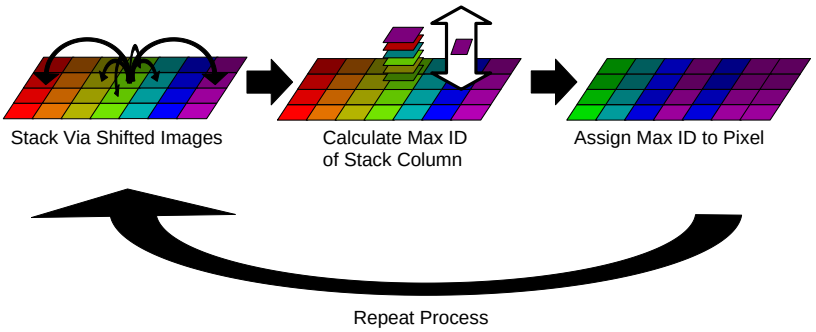

Figure 5.9.: **Cluster ID Assignment via Channel-wise Maximum Operation.**. The ID values of the *ID-tensor* are maxed along the stacked axis in order to assign the maximum ID of all connections to the target pixel. The process is repeated multiple times to connect all points of each cluster.

5.3.2 Lidar Image Panoptic Segmentation: Evaluation

In this section the four different combinations of each the two methods for semantic and instance segmentation are evaluated and compared to each other as well as current state-of-the-art panoptic segmentation methods. The *SemanticKITTI* dataset was used for the evaluation of the presented method. This dataset offers an objective evaluation for testing panoptic segmentation methods: there is a test dataset whose labels are not available to the public. The evaluation server has access to these hidden labels and can thus objectively evaluate the quality of the results.

The core metric of the data set is the *Panoptic Quality* as described in Equation 2.32 on page 23. In the following, however, we extended the attention to the inference speed of all panoptic methods. The *FLIC* algorithm of Chapter 3 has not only shown a higher instance segmentation quality than previous clustering methods, but especially convinced with its fast execution runtime. The possibility of an application such as panoptic segmentation in real time is a basic requirement for operation in a robot or an autonomous vehicle. For this reason, both metrics are highlighted and compared in the following to find the best combination method for each goal as well as the best middle ground for both: the best *Panoptic Quality* that is still real time capable.

The combination of the *FLIC* with only one *Map Connection* and the *MINet* enabled a very fast panoptic segmentation. The total combined runtime of 28.4 ms is many times faster than the acquisition frequency of the sensor. Thus, this combination enables an online capable panoptic lidar segmentation, with a *PQ* of 45.0. This combination is the currently fastest known panoptic segmentation method on the *SemanticKITTI* dataset.

The *Cylinder3D* network shows a very good semantic segmentation metric value (*mIoU*), but it is not real-time capable, it needs more time for the processing of a lidar frame than the recording of the same. The *FLIC* algorithm showed a very high instance segmentation performance in Chapter 3.3.2, using 14 *MCs*. The combination of both methods reaches a *PQ* of 56.9 with a runtime of 208.5 ms.

Table 5.3.: **Panoptic Segmentation Methods on the *SemanticKITTI* Test Set.** The presented combination methods of a retrained *Cylinder3D* [228] and *FLIC* [99] with 14 *Map Connections*, a retrained *MINet* [133] and *FLIC* [99] with one *Map Connection* as well as the same semantic segmentation networks combined with the novel *FLIC++* as described in Section 5.3.1 are compared to state-of-the-art panoptic segmentation networks. The *Scans Per Second* (*SPS*) is given as reported by the respective authors. A *SPS* above ten denotes a real-time capability of the method. The **best**, *second best* and third best result of each column is marked with red bold, blue italic and green underlined text respectively.

Methods	PQ ↑	PQ+ ↑	SQ ↑	PQ(th) ↑	RQ(th) ↑	SQ(th) ↑	PQ(st) ↑	RQ(st) ↑	SQ(st) ↑	mIoU ↑	SPS ↑
Panoptic RangeNet [147]	38.0	47.0	76.5	25.6	31.8	76.8	47.1	60.1	76.2	50.9	11.8
MOPT [109]	43.1	50.7	78.8	28.6	35.5	80.4	53.6	67.3	77.7	52.6	6.9
Panoptic4D (single scale) [36]	50.3	57.8	81.6	45.3	53.2	84.4	54.0	66.8	*79.5*	61.3	-
Panoster [85]	52.7	59.9	80.7	49.4	58.5	83.3	55.1	68.2	78.8	59.9	-✣
Panoptic-PolarNet [232]	54.1	60.7	81.4	53.3	60.6	87.2	54.8	68.1	77.2	59.5	11.6
DS-Net [105]	55.9	62.5	82.3	55.1	62.8	87.2	56.5	69.5	78.7	61.6	-
EfficientLPS [181]	57.4	63.2	*83.0*	53.1	60.5	*87.8*	*60.5*	*74.6*	*79.5*	61.4	4.7
GP-S3Net [164]	*60.0*	**69.0**	82.0	*65.0*	**74.5**	86.6	56.4	70.4	78.7	**70.8**	-
Panoptic-PHNet [132]	**61.5**	*67.9*	**85.7**	**66.9**	*73.3*	**91.5**	**63.0**	**76.1**	**81.5**	*68.4*	-
MINet [133] + *FLIC (1 MC)* [99]	45.0	53.2	77.3	38.7	47.8	80.3	49.5	63.6	75.0	57.4	**37.8** †
Cylinder3D [228] + *FLIC (14 MC)* [99]	56.9	62.9	81.8	54.7	64.1	85.1	58.6	71.4	*79.5*	65.4	4.8 ‡
MINet [133] + *FLIC++*	47.0	55.2	78.2	43.5	52.3	82.7	49.5	63.6	75.0	57.4	*17.3* ✿
Cylinder3D [228] + *FLIC++*	58.0	64.0	82.6	57.2	65.7	86.9	58.6	71.4	*79.5*	65.4	4.7 ❀

✣ The authors of [85] claim a SPS frequency of 58. Due to the fact that their work is based on *KPConv* [15] with a SPS of ~5 it is most likely a decimal error in their report. This was also noticed by the authors of [232]

† *MINet* 16.9 ms ; *FLIC* with one *Map Connection* 9.5 ms
‡ *Cylinder3D* 170 ms ; *FLIC* with 14 *Map Connections* 38.5 ms

✿ *MINet* 16.9 ms ; *FLIC++* 40.8 ms
❀ *Cylinder3D* 170 ms ; *FLIC++* 40.8 ms

The newly presented *FLIC++* algorithm of Section 5.3.1 was combined with the same two semantic segmentation networks. The 18 connections of the *FLIC++* are not listed separately, since no different versions are used here.

The combination of *MINet* and *FLIC++* show a real-time capable panoptic segmentation at a runtime of 57.7 ms with a better *Panoptic Quality* of 47.0 compared to the *FLIC* version. Thus, the *FLIC++* and *MINet* combination needs a little more than half of the acquisition time of a lidar frame for the panoptic segmentation of the same.

The last combination was that of the *Cylinder3D* network with the *FLIC++*. This method achieves a *PQ* of 58.0 and is one of the top three methods for panoptic segmentation together with the two methods *GP-S3Net* [164] (60.0 *PQ*) and *Panoptic-PHNet* [132] (61.5 *PQ*). The authors of *GP-S3Net* have not published their inference runtime, therefor a direct comparison is not possible with the runtime of the presented combination which requires 210.8 ms, a little more than twice the acquisition time of a lidar frame with the present sensor. The authors of *Panoptic-PHNet* report an *SPS* of 11.0 which is real-time capable, but at a batch size of 8 which is not representative for an online application, as opposed to our method which is processed frame-wise to mimic an online inference.

All presented combinations as well as the mentioned state-of-the-art methods are listed in Table 5.3 for the main metrics as well as the *PQ+* which is a modification of the *PQ* metric that uses just the *IoU* for stuff classes without distinguishing between different segments. Furthermore the general *SQ* is listed for each method as well as the *mIoU* and for all stuff and thing classes the separate *PQ*, *SQ* and *RQ*.

The evaluation shows that the combination of different semantic segmentation networks and the presented *FLIC* and *FLIC++* offers a good combination for panoptic segmentation. The method can be arbitrarily adapted to different targets: If a better panoptic quality is desired, this can be achieved by using non-real-time capable networks together with the *FLIC++* algorithm. If, on the other hand, the goal is a real-time application, one can keep the instance segmentation part and replace only the semantic network part with a faster model. Here one has a large margin, since the fastest instance segmentation method, the *FLIC* in its basic configuration creates an additional computation time of only 6 ms as shown in chapter 3.3.1 of page 39.

5.4 Conclusion

This chapter introduces two novel methods for lidar panoptic segmentation, a technique used to label objects in a three-dimensional point cloud based on their class and instance. These methods combine heuristic algorithms and deep learning to achieve highly accurate and efficient segmentation results.

The effectiveness of these new methods is thoroughly evaluated and compared against the current State of the Art in the field of lidar panoptic segmentation. The comparison demonstrates that the proposed methods achieve competitive results in terms of both panoptic quality and runtime efficiency.

The results of this study highlight the potential of these new methods to significantly advance the field of lidar panoptic segmentation, a crucial technology for various applications including autonomous driving, robotics, and surveillance. With further development and refinement, these methods could have broad and impactful applications in the field.

Part II

Enhancing Lidar Segmentation Perception and Adaptability: Novel Techniques for Data Augmentation and Domain Adaptation

6 Developing Advanced Lidar Point Cloud Augmentation Methods for Improved Segmentation

In Chapters 3, 4, and 5 of Part I, new approaches for lidar data segmentation were presented, each with their own advantages and disadvantages. The methodology in Chapter 3 involved an algorithmic approach that used two predefined parameters to separate the ground from upright objects and objects from each other. However, due to its rigidity, this approach was prone to corner cases, such as when it fails to separate two pedestrians walking hand in hand or a car parked very close to a wall. In contrast, deep learning approaches recognize the structure and shape of objects themselves based on a larger number of internal parameters of the neural networks. To train these networks effectively, as described in Chapters 4 and 5, large amounts of data are required to learn the generalized structure that defines a particular type of object.

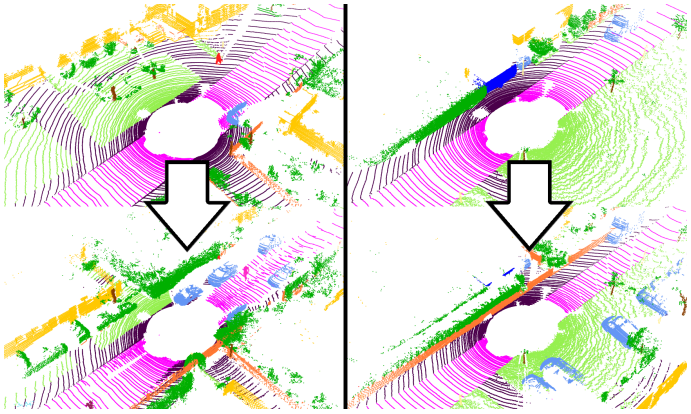

Figure 6.1.: **Two Lidar Point Clouds Enriched via the Structure Aware Point Cloud Augmentation Methods Presented in this Chapter.** The original lidar point clouds (*top*) are augmented with parts of other point clouds to create entirely new scenes (*bottom*) to train neural networks for segmentation.

However, annotating segmentation data is expensive, making it difficult to acquire enough data to train well-performing networks for the task. To overcome this challenge, augmentation methods can be used to modify annotated data to look like new data for the network during the training process. In this chapter, novel augmentation methods are presented that are specifically designed for use in training semantic segmentation networks, but also have applications beyond that for lidar-based neural networks.

The augmentation methods presented in this chapter were created to address two issues with training data for semantic segmentation: overfitting to small data pools and class imbalance. Public datasets, such as *SemanticKITTI* [39] and *nuScenes* [13], as well as closed source datasets, like the internal Aptiv lidar segmentation dataset, often have imbalanced class distributions, with certain classes, such as cars, being more prevalent than others, like motorcyclists.

Despite a number of successful recent approaches for augmentation methods to overcome imbalance and overfitting issues in the two-dimensional image domain [68, 14, 23], the extension of these concepts into the three-dimensional lidar domain creates a domain shift between original data and augmented data [28, 17, 5], as the underlying sensor structure is no longer valid and creates data that a lidar sensor could never capture. Key characteristics of the data, such as scan line integrity, field of view, and object occlusion, are extensively altered by these augmentation methods. Consequently, this makes the use of networks that rely on these structural aspects, such as [7, 61], infeasible. These key characteristics will be called the structure of lidar data in this chapter.

The proposed augmentation techniques of this chapter were designed to address the issues of class imbalance and overfitting in the training data for semantic segmentation of lidar data. The methods preserve the structure of the original raw lidar data while augmenting the content (compare FIGURE 6.1). The augmentation pipeline consists of two novel modules that can be applied to the training of any neural network for semantic segmentation of lidar data. The first method injects additional objects into a lidar point cloud, while the second method combines two point clouds to create a third. The effectiveness of the augmentation pipeline was evaluated using state-of-the-art networks for semantic segmentation of outdoor automotive lidar data. Neural networks from the three most common approaches were used in the evaluation to demonstrate the model independence of the proposed augmentation methods. *KPConv* [15] for the point-wise models, for the range image-based models *RangeNet++* [7], *SalsaNext* [61] and *MINet* [133], and lastly for the voxel-based models *Cylinder3D* [228]. The presented method applied to the training of the later network achieves state-of-the-art performance: it reached the 1^{st} place for "Semi-Supervised Semantic Segmentation" on the *SemanticKITTI* dataset as of December 1^{st} 2022 in the bins 1%, 10% and 50% [1].

The literature research, method description, evaluation, and large parts of the content of this chapter were already published in a conference paper [97]. Some sections, figures and tables are taken in part or in whole from these sources.

The following main contributions are provided in this chapter:

- A novel lidar point cloud augmentation method.
- A thorough evaluation of the method.
- A detailed ablation study that illustrates the influence of all components.
- A comparison with state-of-the-art lidar augmentation methods.

[1] https://paperswithcode.com/sota/semi-supervised-semantic-segmentation-on-24

6.1 Related Work

Neural networks are considered well performing when they generalize to unseen data. When deploying a trained network, one expects a comparable result to that of the validation data that is used to tune the network. Due to overfitting on training data this expectation might not be met. There are various approaches to address this problem. In the following these approaches are separated into model-related and data-related measures.

The first category of overfitting counter measures consists of methods such as *regularization* [95, 208] and *dropout* [185]. The former technique discourages the model from overfitting by imposing a penalty on complexity. The later randomly "drops" parts of the network to prevent co-dependencies among parts of the network.

The second category represents measures that are applied directly to the training data. The overarching goal is to continuously change the data so that overfitting can be prevented. This can also include methods such as over-sampling rare classes [140], expanding the training data with other datasets [4] and pretraining the network on other data domains [9].

In the point cloud domain however, data augmentation primarily relates to the direct modification of the existing training data. Global augmentations are distinguished from local augmentations and context augmentations. The former are applied to entire lidar frames, the second are applied to individual objects and ultimately context augmentations represent the mixture of lidar frames with the context of other frames on a larger scale than limited local augmentations.

6.1.1 Global Augmentations

Lidar point clouds as well as point clouds from other sensor types are usually augmented by applying random translations, rotations, flips and point drops. These techniques show improved generalization of neural networks due to a seemingly larger training dataset [25, 27].

6.1.2 Local Augmentations

In the two-dimensional image domain, different variations of *Copy-Paste* augmentations [23] have been successfully used in various applications. These methods cut out parts of an image and paste them onto another image. Similar approaches are also present in the three-dimensional lidar area. Previous works have already shown the benefits of injecting underrepresented classes in object detection tasks into training points. The most notable example is [216] in which Yan et al. created a database that contains the cut-out point clouds of rare objects using the three-dimensional cuboid annotation around the objects. During training these objects are sampled from the database and injected into the point cloud at random locations. To ensure physically possible locations the cuboid boundaries of injected objects were compared in bird's-eye-view with other object cuboids to avoid collisions [216]. The work of [21] additionally removed all points behind the cuboid of the injected object in polar coordinates to remove any possible overlap with existing objects, thus sidestepping the

collision issue mentioned by [216]. In [108], Hu et al. applied a sub-sampling of the original sampled injection object along the scan lines and between scan lines to extend the distance of injections in the target point cloud, while keeping a similar structure of the lidar scan lines. The authors of [12] used a multi-modal approach, in which they used an instance segmentation network to sample *"foreground"* instances in the camera image. These were projected onto the lidar points to remove occluded points. Lately, injection methods have also been mentioned in lidar segmentation publications [16, 232] which were described similarly to the method in [216]. The most comparable approach to the injection augmentation presented in Section 6.2.2 is [20] in which the authors applied a *Copy-Paste* algorithm [23] to the range image to inject cars in front of other objects, but cancel injections that are injected behind other objects (e.g., other cars).

The injection method presented in Section 6.2.2 is the first to apply a point-wise occlusion competition between target and injection points, thus removing occluded points from the target point cloud but also the injection objects according to their distance from the lidar sensor. This ensures that the lidar sensor structure is retained and the augmented data can not be distinguished from real lidar sensor data.

6.1.3 Context Augmentations

Augmenting the context of an entire scene is another approach to prevent overfitting of a network. Object recognition models and models for semantic segmentation tend to perceive individual objects together with their environment. In doing so, the network can place an excessive focus on the environment which can change on unseen data in a different context [168, 5]. Context augmentation is used to minimize such connections between objects and their context by using individual objects in other scenes (see Section 6.1.2) or by mixing entire scenes.

There are several examples in the image domain where parts of pictures were completely removed [68], parts were cut out and pasted onto other pictures [23], or several pictures were blended together in order to minimize the contextual dependence of the image content [14].

This type of augmentation is also used in the three dimensional domain. The approach presented in [28] blended two point clouds of objects to generate new examples. The work in [17] extended the approach of [23] into the three-dimensional domain and cut and pasted parts of one object point cloud into another to create new mixes and enhance the model robustness against point attacks. The authors of [5] overlayed two full scenes, to create a new combination scene. The authors showed that out-of-context mixing reduces overfitting to the training set which lead to a better generalization on unseen data.

The method for context augmentation presented in Section 6.2.3 is the first to use a sensor-centric approach for the fusion of two point cloud scenes. Thus, opposed to all previous mentioned methods, the lidar sensor structure is kept intact and creates fused scenes which are structurally indistinguishable from real data.

Other augmentation methods that were published after the methods outlined in Section 6.2 put a similar focus on the lidar sensor structure for local and context augmentations. *LaserMIX* [122] focuses on the lidar scan lines as connected components. The authors assume an isotropic arrangement of the scan lines around the sensor. They partition the point cloud along defined ring segments and extract whole scan lines or segments of the same from a point cloud, and combine them with parts of other point clouds. In this way they achieve a pseudo lidar structure of the generated point clouds, but neglect the physical influence of the mixed parts among each other, e.g., in the form of occlusion of distant areas by inserting objects in the near area.

PolarMIX [212] is an augmentation method that uses the rotational structure of 360° lidar sensors as a basic assumption. The authors use a cylinder coordinate transformation to create local and context augmentations for semantic segmentation data of lidar point clouds. They extract pie-like slices from the point cloud to blend them with another point cloud for random angle slices. They also extract the dynamic instances and insert them into other point clouds using a bird's eye view *Copy-Paste* approach. This approach, similar to the previous one, creates point clouds that produce good context mixing but fail to generate the inherent physically limited structure of the lidar sensor.

In Section 6.3.4, these two methods are directly compared with the one described in this chapter for use in improving the training of neural networks for semantic segmentation of lidar point clouds.

6.2 Structure Aware Lidar Augmentation Methods

The main focus of the augmentation methods for semantic lidar segmentation outlined in this section is on preserving the data structure of the sensor. In Chapter 2.1.1, the functionality of rotating lidar sensors was briefly described. While these sensors map a complete 360° three-dimensional point cloud of their environment, they are subject to clear physical limitations, such as the occlusion by objects. Lidar sensors can not "see" through objects. For the sake of completeness, it should be mentioned that some lidar sensors return multiple values per measurement, for example on a window pane, a rain drop or the edge of a solid object, when measuring the depth of a point that is not completely opaque or entirely blocking the light beam. The depth values for this first hit and the object behind it are then returned as separate depth values. For fixed objects that are hit by the entire laser beam these values are twice the same or they only return the single depth information depending on the manufacturer and the user setting of the sensor. For the sake of simplicity, however, these special cases are ignored in the following.

6.2.1 Structure Aware Global Lidar Augmentation Methods

Most global augmentation methods, such as the ones mentioned in Section 6.1.1 keep the previously mentioned lidar structure intact. By globally rotating and flipping the point cloud around the sensor origin, these methods ensure that the scan lines and the line-wise point distances are kept the same relative to each other and to the sensor. No translations to the point clouds are used in the evaluations

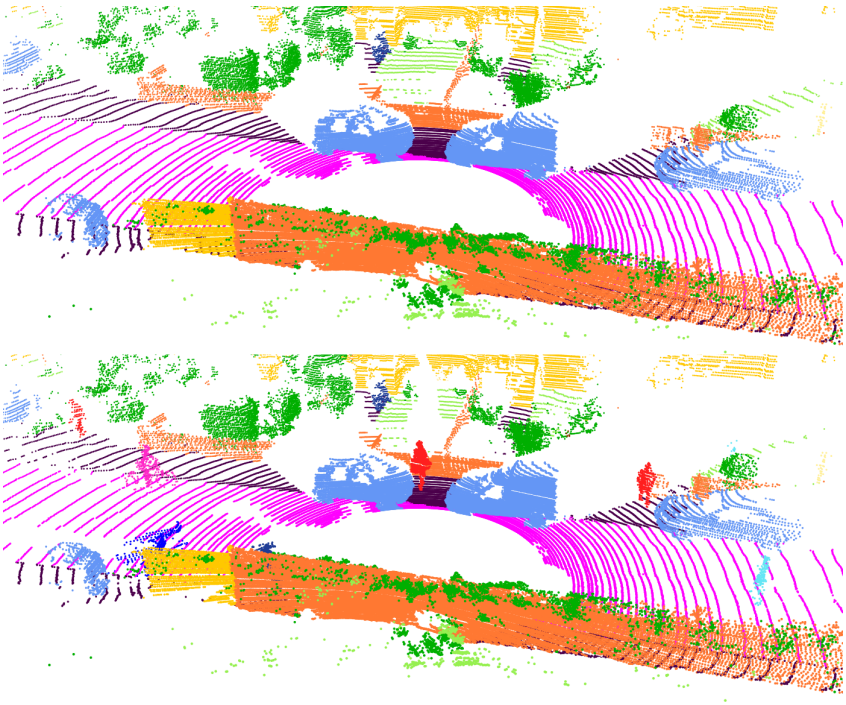

Figure 6.2.: **Instances Injected into a Scene.** *Top*: Original lidar scene. *Bottom*: Augmented scene enriched with additional ●"*pedestrians*", ●"*bicycles*", ●"*bicyclists*" and ●"*other – vehicles*".

in Section 6.3, as they change the point to sensor distance differently for every single lidar point and therefore break the sensor structure.

6.2.2 Structure Aware Point Cloud Injection

The injection method outlined in this section is built on the concepts presented in Section 6.1.2, but extended the sampling and the injection into the range image domain based on the recording method of rotating lidar sensors. The range image is a very close representation of what the lidar sensor is able to capture. The semantic point-wise labels are used to extract rare objects from the training data without including the ground, unlike the methods mentioned in Section 6.1.2. All sampled objects are stored in a database for optimized access. In the injection step objects are sampled from the database and global augmentations (flips, rotations and point drops) are applied to the objects. As these augmentations retain the original lidar structure, the injection object can be projected directly into the range image domain, while changing the location and point density.

Each pixel in the range image of the to-be-injected object is compared with the corresponding pixels of the target point cloud in the range image:

$$\mathcal{S}_{RP,inj} = \begin{cases} 1, \text{ where } 0 < D_{RP,inj} < D_{RP,scene} \\ 0, otherwise, \end{cases} \tag{6.1}$$

where $\mathcal{S}_{RP,inj} \in \{0,1\}$ is a binary mask in the range projection view, that defines, if the pixel position in the range image will be occupied by the injection or the original scene. $D_{RP,inj}, D_{RP,scene} \in \mathbb{R}_+$ are the range image projected point clouds of the injection and the original scene, respectively. The point closer to the sensor wins the range competition and remains in the new point cloud, the farther point is removed. The advantage of this method is, that for injected objects e.g., behind street lights, the occluded points are removed from the injection object by the origin mask $\mathcal{S}_{RP,inj}$ instead of preventing the injection altogether which is different to the methods mentioned in Section 6.1.2. With the same technique the points behind the injected object are removed to imitate the lidar shadow. The objects are not translated in the range image between lidar channels, as this creates floating or submerged objects in the target point cloud.

These two steps - sampling and injecting - enables the enrichment of any training point cloud with additional objects, especially of rare classes. The structure of the point cloud is also kept consistent with what the lidar sensor can capture. Therefore, the created point clouds are much more similar to the validation and test data than other naive injection methods which create a noticeable difference between training and test data. Figure 6.2 shows the direct comparison of a training point cloud for semantic segmentation and the same point cloud augmented by the presented structure aware point cloud injections. "*Bicycles*" denotes free-standing bicycles without a rider, while the "*bicyclists*" label is assigned to bicycles with a rider.

In order to inject underrepresented classes, the shown approach compares the present class distribution in the target point cloud with a desired distribution. Additionally a parameter is defined for a maximum number of injections. The process is started by randomly choosing an injection class: if the given class is already present in the target point cloud, in a proportion larger than the desired amount, the process switches to a different randomly chosen injection class. The random sampling, checking and injection of objects into the target point cloud continues until either the desired class distribution is reached, or alternatively the maximum number of injections.

6.2.3 Structure Aware Point Cloud Fusion

The second augmentation module in this chapter goes beyond the injection of single objects into an otherwise unchanged point cloud. Instead of a single object, this method uses a second training point cloud and applies global structure-maintaining augmentations to it. To ensure a valid point cloud, the second cloud is only rotated in increments of the horizontal sensor resolution and a translation of the points is prevented. No scaling is applied to the point cloud, but random flips in the x and y direction are allowed. These restrictions keep the structure of the second point cloud within the lidar sensors recording capabilities.

Based on the range competition technique mentioned in Section 6.2.2, both point clouds are projected into the range image domain and compared pixel-wise via Equation 6.1. The closer point is kept by the binary origin mask $\mathcal{S}_{RP,inj}$, and the farther one is discarded in order to generate a new scene which results in a structurally as well as semantically valid point cloud.

A rotation limitation of $\pm 10°$ is used, to keep the probable direction along the street valid: most recordings of automotive lidar data have a road at the front and back, while the sides are obstructed by

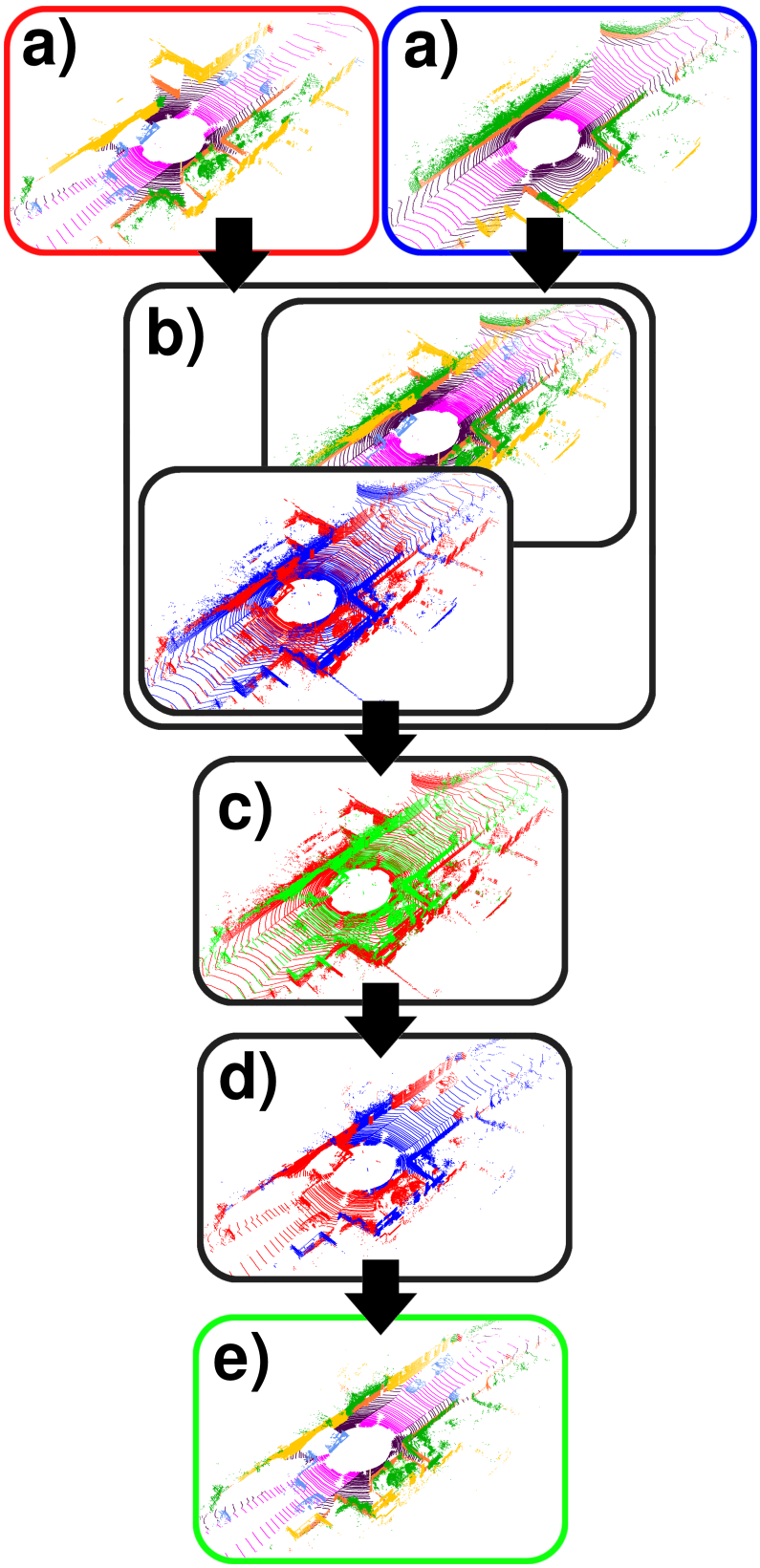

Figure 6.3.: **Visualization of the Point Cloud Fusion Method.** **a)** Two separate independent point clouds from the training set. **b)** Both point clouds overlayed on top of each other. **c)** Range competition of both point clouds: The green points are closer to the lidar sensor. **d)** Fused point cloud of the closer points of both parent point clouds. **e)** Final fused point cloud, exhibiting parts of each parent point cloud.

e.g., houses and parking cars, this limitation of the rotation is chosen to stay close to this orientation for the generated fused point cloud. Figure 6.1 and Figure 6.3 e) show examples of generated scenes. The global augmentations (i.e., rotations, flips and point drops) applied to the second point cloud lower the likelihood of ever merging the same point clouds in the same arrangement, to a negligible probability.

Figure 6.3 visualizes the process to generate a point cloud consisting of parts of the two parent point clouds. While the lidar data structure is preserved, there is a risk of objects from both point clouds merging and thus causing an unrealistic representation. It can also happen that nonsensical scenes are created, such as a freeway guardrail that blocks a house entrance, or a tree that protrudes from the roof of a car. However these nonsensical, out-of-context fusions improve the networks as will be shown in Section 6.3.1. In order to keep the parameters of the injection module valid, the fusion step is applied before the injection step in the full pipeline.

6.3 Evaluation

The proposed augmentation methods were added to the training data-loaders of different neural networks on the dataset *SemanticKITTI* [39] for most evaluations in this section. These networks were trained on the annotated training sequences $0-7$ and $9-10$, and validated on the annotated log 8. The injection and fusion methods only sampled data from the 10 training logs to ensure a fair comparison and prevent the usage of validation data in the training pool.

The presented augmentation methods are not limited to a special kind of network, but can be added to point-based, voxel-based and also range image-based networks without any issue, as the generated point clouds exhibited the same structure and orientation as real raw data. In [19] a method was proposed to re-engineer the lidar channel indices, that was also used in the following experiments to improve the range image projection for the augmentation process. For all networks the same augmentation parameters were used:

- 50% probability to apply global augmentations
- 30% probability to mix with a second random scene
- 50% probability to inject instances
- maximum of three injected instances per frame
- desired share of 2% for all injection classes

Point-based Networks

KPConv [15] was selected to represent point-based segmentation methods for the use with the augmentation methods of this chapter. The decision fell on this network as it is the best performing published point-wise network on the *SemanticKITTI* [39] leaderboard. The *Pytorch* implementation by the original authors[2] was used, with slight modifications to the training parameters to fit the training and validation loop into the available GPU, namely the input radius of each ball queries was increased to 51 meters and the size of the first sub-sampling grid increased to 0.2 m. The other

[2] https://github.com/HuguesTHOMAS/KPConv-PyTorch

parameters of the network were left as the authors provided them. These hardware-based limitations lead to a decline in performance of the network on small classes such as bicycles and pedestrians as can be seen in Table 6.1.

Voxel-based Networks

As the representative for voxel-based networks the public repository[3] of the *Cylinder3D* [228] network was used which was published by the original authors. To speed up the training time, a *one cycle learning rate scheduler* [182] was added to the training loop. All other parameters were kept as provided by the authors.

Range Image-based Networks

For the last group three open source networks were used: *Rangenet++* [7][4], *SalsaNext* [61][5] and *MINet* [133][6]. These network types are based on the lidar structure of the point clouds and require a consistent structure to function properly. Therefore three networks were used to showcase the structure retention of the augmentation pipeline. No parameters of these three networks were changed, and the training scripts were used as provided by the authors.

Table 6.1 presents a direct comparison of the provided checkpoints and the same models retrained with the augmentation pipeline discussed in this chapter. To ensure a fair comparison, both the baseline and augmentation versions of *KPConv* [15] were trained from scratch, as a pretrained checkpoint was not available. The results demonstrate that the augmentation pipeline significantly improved the performance in all three network categories, without any changes to the network or hyperparameters.

The three range image based networks *RangeNet++*, *MINet* and *SalsaNext* improve by 3.5, 7.8 and 4.3 *mIoU* respectively. The *MINet* retrained on the proposed augmentation pipeline outperforms the original network on all 19 classes without the use of the *edge-loss* that was proposed in [133].

The point-based *KPConv* network improves by 2.6 *mIoU*. The most improvement can be seen for the classes "motorcycle" with 11.5 *IoU* and "person" which jumps from an *IoU* of 0.0 to 32.1.

Finally, even though the baseline *Cylinder3D* model had already been trained with various data augmentations, such as instance injections, rotations, and scaling, the presented method managed to further enhance its performance. Notably, it significantly improved the performance of underrepresented classes such as "bicycle," "motorcycle," and "person," with improvements in *IoU* of 4.1, 4.9, and 5.3, respectively. Although the overall improvement in *mIoU* was only 1.0, the presented method improved the model's accuracy especially for challenging classes.

[3] https://github.com/xinge008/Cylinder3D
[4] https://github.com/PRBonn/lidar-bonnetal
[5] https://github.com/Halmstad-University/SalsaNext
[6] https://github.com/sj-li/MINet

Table 6.1.: **Results of Various Networks with and without the Presented Augmentation Pipeline on the *SemanticKITTI* Validation Set.** The augmentation pipeline of this chapter is abbreviated with the designation *SAPCA* for **S**tructure **A**ware **P**oint **C**loud **A**ugmentation.

Methods	mIoU ↑	car	bicycle	motorcycle	truck	other-vehicle	person	bicyclist	motorcyclist	road	parking	sidewalk	other-ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign
RangeNet++ [7]	52.8	91.0	25.0	47.1	40.7	25.5	45.2	62.9	0.0	93.8	46.5	**81.9**	0.2	85.8	54.2	**84.2**	52.9	**72.7**	**53.2**	**40.0**
+ *SAPCA*	**56.3**	**91.4**	**34.1**	**56.8**	**57.5**	**39.7**	**53.4**	**66.8**	**6.2**	**94.4**	**49.9**	81.6	**0.4**	**87.0**	**58.8**	83.1	**53.0**	71.6	45.3	37.8
MINet [133] †	52.8	89.6	36.8	36.5	56.2	31.9	55.4	67.1	0.0	91.5	33.1	76.4	0.8	83.1	42.5	84.7	57.1	71.2	50.5	39.6
+ *SAPCA*	**60.6**	**90.6**	**50.9**	**63.1**	**75.6**	**46.1**	**71.1**	**87.4**	**2.0**	**93.1**	**38.9**	**79.4**	**2.5**	**87.4**	**45.8**	**83.8**	**62.2**	**70.1**	**57.0**	**44.8**
SalsaNext [61]	56.9	86.7	40.7	42.0	**79.3**	42.5	64.6	69.4	**0.0**	94.5	**42.6**	**80.2**	3.5	80.6	48.3	80.8	61.6	65.1	53.1	44.9
+ *SAPCA*	**61.2**	**90.4**	**46.2**	**58.7**	66.4	**54.1**	**71.9**	**80.1**	**0.0**	**94.6**	42.1	**80.2**	**3.7**	**87.4**	**50.7**	**86.0**	**67.1**	**72.5**	**62.1**	**48.4**
KPConv [15]	43.3	93.7	**0.0**	0.9	79.6	20.6	0.0	**0.0**	**0.0**	90.4	20.7	71.7	**0.0**	89.5	54.9	**86.2**	51.7	**70.2**	55.9	**37.3**
+ *SAPCA*	**45.9**	**93.9**	**0.0**	**12.4**	**80.9**	**22.3**	**32.1**	**0.0**	**0.0**	**90.8**	**23.5**	**72.8**	**0.0**	**89.6**	**55.7**	85.4	**54.7**	68.3	**56.6**	32.3
Cylinder3D [228] ‡	66.9	97.1	54.5	80.9	**85.1**	70.3	76.5	92.2	**0.0**	94.6	44.8	81.2	**1.0**	**90.5**	**58.7**	86.6	**70.8**	70.5	64.2	**51.8**
+ *SAPCA*	**67.9**	**97.3**	**58.6**	**85.8**	84.8	**71.4**	**81.8**	**92.8**	0.8	**94.9**	**45.3**	**81.8**	0.4	90.2	58.3	**87.6**	69.0	**72.6**	**65.8**	51.5

† The basic MINet without a *k*-NN post-processing was used for a better evaluation of the network.

‡ The used checkpoint has already been trained with various data augmentations, such as instance-level rotation and scaling and therefore performs better than noted in the original paper. See `https://github.com/xinge008/Cylinder3D/issues/88`

The class-wise comparison revealed a significant overall improvement in all networks, particularly for the underrepresented dynamic object classes. This can be attributed to the augmentation methods, which encourage the networks to encounter these classes more frequently through the injection technique. Furthermore, the fusion module mixed other classes out of their original context, resulting in improvements for most classes, even those that were not explicitly injected. By focusing on the underlying capabilities of the lidar sensor, the method enabled the augmentation of lidar data while preserving its structural integrity. Consequently, this approach can be applied to various semantic segmentation models for lidar data, as the augmented point clouds maintain the same structural properties as the original raw data.

6.3.1 Improving Segmentation

SemanticKITTI Dataset

The *Cylinder3D* that was retrained on the augmentation pipeline of this chapter was inferred on the hidden test set of SemanticKITTI [39]. The results are directly compared to the provided checkpoint of the official repository. Please note that no test time augmentations or hyperparameter tuning was applied to the two listed results in Table 6.2 . It is also worth mentioning, that the open sourced *Cylinder3D* repository was not the complete network described in [228], but a reduced version[7]. As this chapter is mainly concerned with the presented augmentation methods and not the capabilities of the used network, the missing parts of the model were not re-implemented, and the code was used as provided by the authors. Table 6.2 shows, that the model retrained with the augmentations outperforms the original checkpoint on 14 out of 19 classes, marked with bold text in the table, while they are on par for one more class. The *mIoU* of the *Cylinder3D* model was improved by 1.5 points from 63.9 to 65.4, solely by the augmentation pipeline of this chapter.

Furthermore, these results imply, that the generalization of the network further improves from the augmentation methods. The offset between the test data and the validation data is larger for models, that are tuned to the validation data, as a bias is introduced by using the best checkpoint for a given validation set.

The original *Cylinder3D* checkpoint inferred on the test data reaches 95.52% of the performance when inferred on the validation set as seen in Table 6.1. The augmentation methods of this chapter improve the performance on both the validation and the test data, but they also narrow the gap between the two, due to which the model trained on augmented data reaches 96.31% of the validation performance on the test set. Less bias towards the validation data is introduced and the model is truly generalizing better due to the proposed augmentation methods.

Aptiv Internal Dataset

The same network and augmentation pipeline was used with a different lidar dataset. The internal Aptiv dataset for lidar segmentation is built on the *Hesai Pandora* sensor [103]. The entire dataset

[7] https://github.com/xinge008/Cylinder3D/issues/23

Table 6.2.: **Results of *Cylinder3D* with and without the Presented Augmentation Pipeline on the Hidden *SemanticKITTI* Test Set.** The augmentation pipeline of this chapter is abbreviated with the designation *SAPCA* for **S**tructure **A**ware **P**oint **C**loud **A**ugmentation.

Methods	mIoU ↑	car	bicycle	motorcycle	truck	other-vehicle	person	bicyclist	motorcyclist	road	parking	sidewalk	other-ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign
Cylinder3D [228]	63.9	96.7	60.3	57.4	43.2	49.6	70.0	65.1	**12.0**	**91.6**	64.6	**76.0**	**24.3**	90.0	63.4	**84.8**	70.7	67.5	62.1	64.0
+ *SAPCA*	**65.4**	**96.8**	**61.4**	**63.1**	**54.0**	**58.7**	**71.3**	**67.3**	11.7	90.8	**66.0**	74.3	12.0	**90.6**	**64.7**	**84.8**	**73.0**	**68.4**	**64.5**	**69.1**

Table 6.3.: **Results of *Cylinder3D* Networks with and without the Presented Augmentation Pipeline on the Aptiv Validation Set.** The augmentation pipeline of this chapter is abbreviated with the designation *SAPCA* for **S**tructure **A**ware **P**oint **C**loud **A**ugmentation.

Methods	mIoU ↑	Car	Truck	Bike	Person	Guardrail	Road	Overdrivable	Underdrivable	Nondrivable
Cylinder3D [228]	64.7	89.4	62.6	32.6	71.1	58.6	86.4	54.2	33.3	94.1
+ *SAPCA*	**71.3**	**90.7**	**76.5**	**51.1**	**73.7**	**63.5**	**88.0**	**67.4**	**35.6**	**95.4**

contains 4,502 panoptically annotated frames. When compared to the *SemanticKITTI* dataset [39] with its 23,201 panoptically annotated frames, the difficulty of training networks on this data and the importance of effective augmentation methods becomes apparent. The dataset was divided into 3,875 training samples and 627 frames for validation, for the purpose of training the semantic segmentation network *Cylinder3D* [228]. The same network was trained twice from scratch with the same parameters. As shown in In Table 6.3, the network trained with the augmentation methods outperforms the original network trained on the raw data for every single class. The *mIoU* improved by a total of 6.6 points representing a relative improvement of 10.2%.

6.3.2 Ablation Study

An ablation study of the individual augmentation components listed in Section 6.2 was conducted with the *Cylinder3D* model using the *SemanticKITTI* dataset. The main focus of this study was to understand the impact of each part of the augmentation pipeline on the final performance of the network. The baseline in Table 6.4 was a basic *Cylinder3D* network retrained from the original checkpoint without any augmentations. The validation loop started after the first 10 epochs and ran for a total of 30 epochs to measure the performance. Global augmentations improved the final *mean Intersection over Union* (*mIoU*) score only slightly from 65.18 to 65.73 (+0.55). A more significant increase was observed in the final segmentation quality by adding either the fusion module reaching an *mIoU* of 67.29 (+2.11) or the injection module with 66.97 (+1.79). The best results were achieved by combining all methods which resulted in an *mIoU* of 67.92 (+2.74). It is worth noting that using only the fusion method performed better than just the injection method. This appears to be due to the recombination of all classes, rather than just the dynamic subset on which the injection method produces better results.

Table 6.4.: **Ablation Study using the *Cylinder3D* Network on the *SemanticKITTI* Validation Set.** All models are trained from the original checkpoint with the same parameters.

Baseline	Global Augs.	Inject	Fusion	mIoU ↑
✓				65.18
✓	✓			65.73
✓	✓	✓		66.97
✓	✓		✓	67.29
✓	✓	✓	✓	67.92

Table 6.5.: **Performance on Reduced Data Sizes.** *Cylinder3D* models trained from scratch on artificially reduced subsets of the training data.

Data	Baseline	+Augmentations	$\Delta abs.$	$\Delta rel.$
100%	60.36	67.16	+6.80	+11.27%
50%	57.55	64.94	+7.39	+12.84%
10%	53.90	63.98	+10.08	+18.70%
1%	37.32	50.86	+13.54	+36.28%

6.3.3 Overcoming Data Scarcity

The augmentation pipeline creates novel scenes by merging parts of the existing training data. To evaluate its benefits, it was compared to the addition of real annotated lidar data. The size of the *SemanticKITTI* dataset was reduced by selecting every second, tenth, and one hundredth data sample in a uniform manner from the scene and injection databases. Four *Cylinder3D* [228] networks were trained from scratch on the reduced subsets, with and without the augmentation pipeline, and evaluated on the complete validation set. As shown in Table 6.5, the augmentation methods improved the performance on all data subsets. The networks trained with the augmentation pipeline demonstrated significantly better results compared to the baseline networks trained on the same reduced dataset. Moreover, the augmentation pipeline provided a greater absolute and relative performance offset for smaller subsets of data. Notably, the model trained using the augmentation pipeline with the same parameters and only 10% of the original data outperformed the baseline trained on the full dataset by 3.62 *mIoU*. This suggests that the scenes generated by the augmentation methods are more effective than labeling ten times more data.

6.3.4 State of the Art

Two recently published augmentation pipelines can be compared to the one presented in this chapter. Table 6.6 shows a comparison between the augmentation pipeline presented here and the performance boost achieved by *LaserMIX* [122] and *PolarMIX* [212] on a *Cylinder3D* model for semantic segmentation of the *SemanticKITTI* dataset.

The authors of both publications reported their results on a 10% and 50% subset of the dataset, therefore it is not possible to make a full comparison of the final performance. Nonetheless, the presented method achieved a more significant improvement of 64.0 *mIoU* on the 10% subset than the other

Table 6.6.: **Comparison of the Presented Augmentation Methods to Current State-of-the-Art Lidar Augmentation Methods.** *Cylinder3D* models trained from scratch on full and artificially reduced subsets of the training data. The performance of the presented augmentation pipeline was compared to the two state-of-the-art lidar augmentation methods *PolarMIX* [212] and *LaserMIX* [122] trained on reported sets and subset of the SemanticKITTI dataset. The augmentation pipeline of this chapter is abbreviated with the designation *SAPCA* for **S**tructure **A**ware **P**oint **C**loud **A**ugmentation.

Method	SemanticKITTI [39]			
	1%	10%	50%	100%
Cylinder3D [228]	37.3	53.9	57.6	60.4
Cylinder3D [228] + *LaserMIX* [122]	50.6	60.0	62.3	-
Cylinder3D [228] + *PolarMIX* [212]	-	62.5	-	-
Cylinder3D [228] + *SAPCA*	**50.9**	**64.0**	**64.9**	**67.2**

methods, which achieved 60.0 and 62.5 *mIoU*, respectively. Additionally, the method outperformed *LaserMIX* [122] on the 50% subset, achieving an *mIoU* of 64.9 compared to *LaserMIX*'s 62.3. Based on the reported results, it can be inferred that the presented method performs better on the entire dataset than the other two methods.

As of December 1st, 2022, when used with a *Cylinder3D* model on reduced data pools, the presented method holds the top position for "Semi-Supervised Semantic Segmentation" on the *SemanticKITTI* dataset for the 1%, 10%, and 50% bins [8].

6.4 Conclusion

This chapter proposes augmentation techniques to address imbalanced datasets for semantic segmentation networks. The augmentation pipeline is effective in improving generalization abilities, even with limited data. The approach retains a structure similar to real lidar sensor data, making it compatible with various neural network designs. Evaluations presented in Section 6.3 show that the proposed augmentation methods significantly improve the performance of multiple lidar semantic segmentation models, regardless of dataset size or network design. Furthermore, the proposed method outperforms the current state-of-the-art for semi-supervised lidar semantic segmentation, without the need for additional annotated training data. This is a remarkable accomplishment, considering that data annotation is both time-consuming and expensive. Additionally, the proposed method yields more significant performance improvements for semantic lidar segmentation than a ten-fold increase in additional annotated training data.

[8] https://paperswithcode.com/sota/semi-supervised-semantic-segmentation-on-24

7 Enhancing Lidar Domain Adaptation for Robust Semantic Segmentation

In the previous chapter, novel methods for lidar augmentation were introduced and evaluated to improve the performance of semantic segmentation networks for lidar data by adding more objects and variety to the training data. These augmentation techniques are highly effective in addressing class imbalance and increasing training data diversity. However, augmentations alone may not be sufficient for achieving well-performing semantic segmentation networks beyond a certain dataset due to the significant variation of lidar data across different domains, such as different sensor types [30, 166, 42, 60], environments [121], or seasons [50]. Networks trained on one domain may not generalize well to another, resulting in poor performance on unseen data.

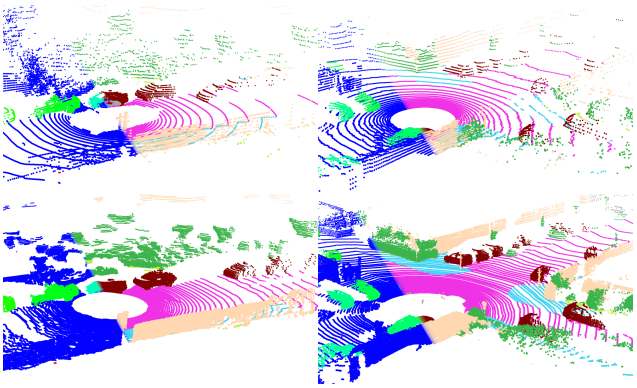

Figure 7.1.: **Panoptic Lidar Point Clouds and Their Respective Twins in a Different Lidar Sensor Domain.** The lidar structure and the existing classes of both datasets was modified so that the scenes existed in both domains: Real nuScenes (top left) as synthetic SemanticKITTI (bottom left), real SemanticKITTI (bottom right) as synthetic nuScenes (top right).

To address this issue, domain adaptation techniques were proposed to bridge the gap between different domains and improve the generalization of semantic segmentation networks for lidar data. Domain adaptation refers to the process of adapting a model trained on one domain to perform well on another domain, without the need for an entirely new annotated training dataset.

Current state-of-the-art domain adaptation methods for lidar segmentation use alignment of geometric and feature statistics at the data level [30, 166], and use network-specific adaptations at the model level to reduce the domain shift between datasets [42, 60].

In this chapter, a new method for domain adaptation is presented that was published in a conference paper [98]. The method exclusively aligns different lidar domains at the data level using sensor-aware domain adaptation modules and data fusion methods that are self- and semi-supervised. By combining point clouds into a static mesh and ray tracing the mesh with a virtual target lidar, the source data is recreated in the structure of the target sensor, as shown in Figure 7.1. These techniques,

along with self- and semi-supervised methods, effectively reduce the domain shift between datasets and enable the training of effective lidar segmentation networks.

The following main contributions are provided in this chapter:

- A novel lidar domain adaption method.
- A thorough evaluation of the proposed semantic segmentation domain adaption method.
- A detailed ablation study that illustrates the influence of all components.
- A comparison to state-of-the-art lidar semantic segmentation domain adaptation methods.

7.1 Related Work

Pretraining and fine-tuning are techniques commonly used in domain adaptation to enhance a model's performance on a specific task or domain. The approach involves training the model on a large, general-purpose dataset before fine-tuning it on the specific domain or task [51, 187]. This helps the model to better understand the unique characteristics and nuances of the target domain or task, resulting in improved performance [189, 47].

Unfortunately, the technique described does not work as well for lidar segmentation [171], therefor other approaches have been explored. In general, domain adaptation for lidar semantic segmentation can be divided into two large categories: *'simulation-to-real'* and *'real-to-real'* domain adaptation.

The *'simulation-to-real'* methods create large pools of annotated training data for a target sensor with a computer program to simulate the sensor data [72]. While this approach can generate a large amount of data, trained networks suffer from a "domain shift" when applied to real data, as simulated environments are too smooth and clean compared to the real world. To address this issue, some researchers have proposed data-level methods to adjust the appearance and sparsity of simulated point clouds to be more similar to real recordings [213, 226], or have added pseudo-labeled real data to simulated data [169]. In addition, simulation environments are constrained in their ability to generate diverse scenarios due to the limited number of pre-designed building blocks available [72].

Several approaches have been proposed for *'real-to-real'* lidar domain adaptation, in which the source domain data are real recordings of a different sensor. These approaches include translation and removal of lidar channels [30], summarization and mesh filling of point clouds [128, 42], surface completion using Poisson surface reconstruction and ray tracing [219], in-painting of sparse labels [113], and use of generative adversarial networks [60] and range image masking [166] to make one dataset look like another.

Previous domain adaptation approaches for lidar semantic segmentation have been limited to specific data structures [166, 60] or have resulted in rough target point clouds with limited details which prevents precise segmentations [128, 42, 219, 113].

This chapter proposes a novel domain adaptation method that combines unsupervised domain adaptation with fusion techniques of self-supervised pseudo labels. By utilizing minimal annotations, the method achieves competitive results in the target domain and overcomes the limitations of previous approaches.

7.2 Lidar Domain Adaptation for Segmentation

In the following section a data-centric approach is outlined for panoptic lidar domain adaptation that preserves semantic and instance labels of the source dataset. The proposed method recreates the scene from the source dataset into a three-dimensional point cloud that matches the shape, range, and structure of any other lidar sensor. This enables the training of various segmentation networks on the generated data. To minimize the domain shift between the generated data and the real data of the target sensor, dynamic objects from the target sensor are incorporated into the static scenes. This is done using either a small pool of annotated data or pseudo-labeled data from previous iterations of trained networks.

7.2.1 Non-Causal Data Collection

To generate a denser representation of real-world scenes captured and annotated in a source dataset, the points of sequential scenes are accumulated over their entire sequence. Both the SemanticKITTI [39, 87] and nuScenes [13, 26] datasets provide ego-motion ground truth that is used for this purpose. To prevent dynamic objects such as moving cars and pedestrians from appearing multiple times in the static point map, all dynamic instances are removed from the point scenes. The resulting scene point clouds appear denser, but the points are still zero-dimensional objects in a three dimensional world (as shown in Figure 7.2 b), i.e., they fill no volume. To sub-select or ray trace the scene point cloud using the structure of the target lidar sensors, methods such as closest-point sampling can be used. However, these methods introduce unrealistic representations, such as visible points behind walls or other objects, due to the lack of direct occlusions [128]. These gaps are instead filled with a mesh representation derived from the scene point cloud in order to create a three dimensional representation of the sequence.

7.2.2 Lidar Mesh Creation

Recreating surface models from point clouds has been studied for almost a century [66], and various methods have been developed, including *alpha shapes* [74], *truncated signed distance functions* [63], and the *Poisson surface reconstruction algorithm* [116]. The latter is used in the presented method.

Poisson surface reconstruction is a method used to reconstruct a watertight, triangulated approximation of an unknown model's surface, based on a set of samples that lie on or near the surface of the

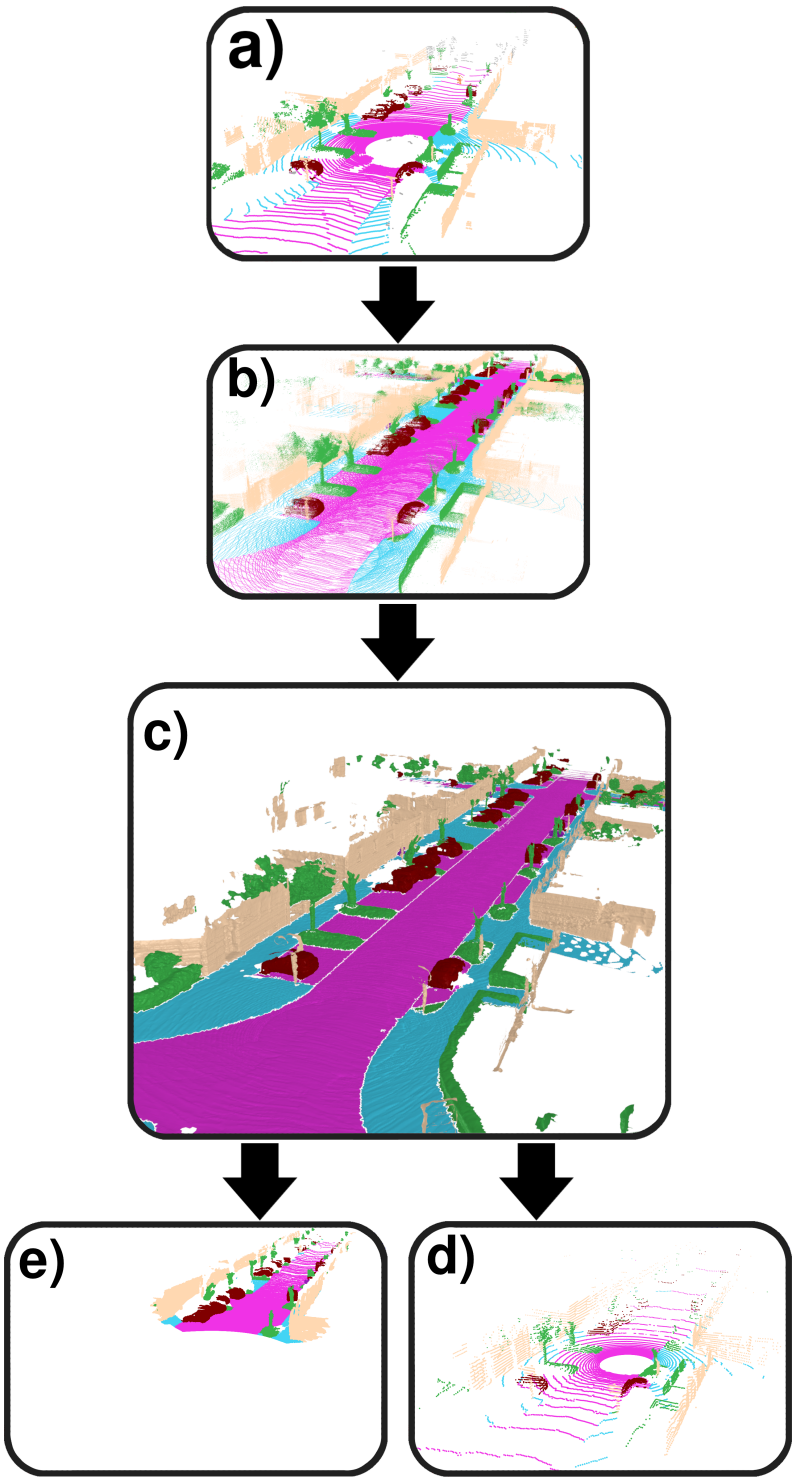

Figure 7.2.: **Restructuring of a Single Dataset in the Form of Two Different Sensors.** The *SemanticKITTI* dataset (a) was summed up for all point clouds in a sequence (b), a mesh world was created (c) and finally retraced in the lidar structure of the *VLP-32C* (d) used in the *nuScenes* dataset as well as the *InnovizTwo* lidar sensor (e).

model. These samples contain a point and an inward-facing normal. The algorithm works by approximating the indicator function of the model and extracting the isosurface as a three-dimensional mesh model. For further information, please refer to the original publication on "*Poisson Surface Reconstruction*" by Kazhdan and Hoppe[116].

The *Open3D* [229] implementation of the *Poisson surface reconstruction algorithm* is used in the following to recreate the scene point cloud as a mesh object. For each mesh vertex the ten nearest neighbors in the original scene point cloud are chosen via k-nearest neighbors sampling [81]. The most frequent values for the class and instance labels are assigned to the vertex of the mesh surface. The intensity value reflects the mean value of the ten nearest original points, with an inverse linear distance weighting.

In Chapter 2.1 the general lidar equation for the remission (compare Equation 2.10) was broken down to the two main influences: reflectance of the target and environmental properties. The environmental properties can not be recreated in the given mesh model, as the volume between surfaces and the sensor is not modeled. Further, the reflectance is based on the reflectivity of the target surface and the inclination angle of the laser beam. While the latter can be modeled from the surface normals of the mesh recreation, the surface properties are too diverse to recreate in an efficient manner [194]. As a result, a choice was made to allocate predetermined intensity levels to each vertex based on the ten closest original points, using inverse linear distance as the weighting factor.

7.2.3 Virtual Lidar Sampling

The point cloud of the mesh object in the structure of the target lidar sensor is recreated using a simplified ray casting method. The mesh environment is projected from Cartesian to spherical coordinates. The mesh transformed to the spherical coordinates is recorded as a depth image from the perspective of the lidar sensor. The camera's location and rotation are adjusted to match the target sensor. The render resolution is chosen as three times the lidar resolution and subsampled to the target sensor's resolution. This decreases the discretization effects at longer ranges, which would cause the effect of "growing" objects at farther ranges. The depth, azimuth, and elevation angle of each pixel are re-transformed into the Cartesian coordinate system to obtain a pseudo lidar point cloud in the structure of the target sensor. The semantic, instance, and reflection values from the mesh model are assigned from the mesh faces to the newly created points. This allows for the structure of any number of different lidar sensors to be recreated using a single mesh world, as shown in Figure 7.2.

7.2.4 Instance Injections

The previously outlined method successfully captures the source data's content in the target data structure. However, it only portrays the stationary components of the source data in the resulting scenes. To overcome this limitation, a semi-supervised approach is utilized to reintroduce dynamic objects into the otherwise empty scenes. General purpose object detectors [127, 178] are applied to

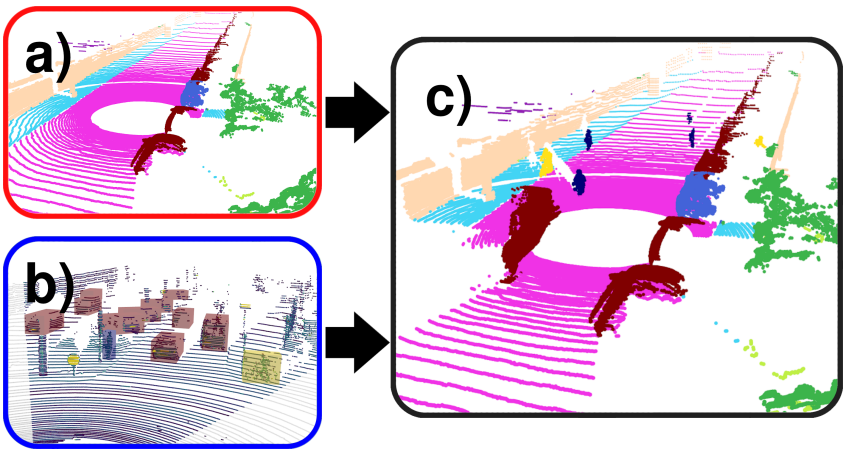

Figure 7.3.: **Target Domain Data Injection into Generated Lidar Point Clouds.** The generated static scene (a) was combined with sampled target sensor (pseudo) ground truth data (b), that was extracted from cuboid labels or alternatively bounding box predictions. The instances were injected into the generated scenes to create dynamic lidar data (c) consisting of parts of the source and the target domain.

the unlabeled target lidar data. The points within the box predictions, along with their semantic and instance labels, are cut out and inserted into the recreated segmentation scenes as dynamic objects. This injection process is shown in Figure 7.3. The insertion is done via the structure aware point cloud injection method described in Chapter 6.2.2 for lidar augmentation of data from the same domain. Another option is to employ the injection method using limited sets of ground truth cuboid or segmentation labels, provided they are accessible for the target data.

The injection technique provides three key benefits. Firstly, it reintroduces dynamic objects into the otherwise static scene. Secondly, it balances the distribution of underrepresented classes, thus increasing the exposure for segmentation networks. Finally, by combining real instance point clouds with generated scene point clouds, it effectively reduces the gap between the real and generated domains.

For the semi-supervised approaches in Section 7.3.1 and 7.3.2, a subset of the provided bounding box labels from the *KITTI* [87] and *nuScenes* [13] datasets, respectively, are used for instance injection.

7.2.5 Mixing Domains

Recently multiple lidar augmentation methods have been published that go beyond injecting single objects into a scene. They attempt complete mixtures between two lidar point clouds recorded at different positions and times. The straightforward concatenation of two point clouds, as proposed by *Mix3D* [5], is used to break up the context of certain classes and objects. A different approach, was presented in Chapter 6.2.3 which keeps only parts of each point cloud according to their distance to the lidar sensor. This creates a mixed point cloud while maintaining the structure of the lidar sensor. The domain mixing approach of this section is based on the latter method and combines synthetic generated scenes with a subset of target lidar data, as shown in Figure 7.4.

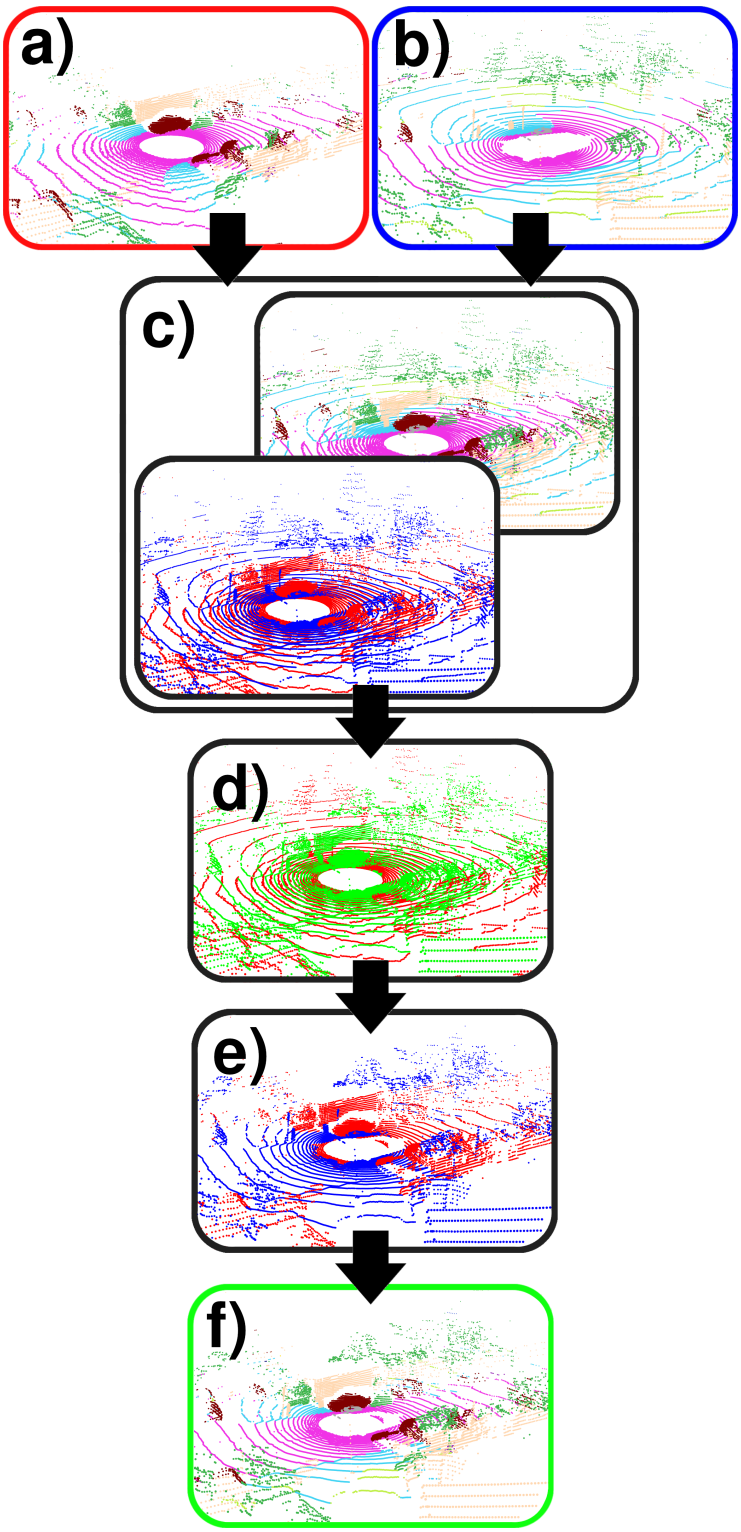

Figure 7.4.: **Point-wise Domain Fusion by Range.** A generated lidar scene (a) and a (pseudo) ground truth lidar frame of the target sensor (b) were selected at random. Both frames are moved to the same origin (c) and a point-wise range competition (d) in the range image domain was applied: the green points were closer to the lidar sensor. A new point cloud (e) exhibiting parts of the real target data (blue) as well as the labeled generated data (red) emerges. The final result (f) is a structurally intact point cloud consisting of both generated and real data.

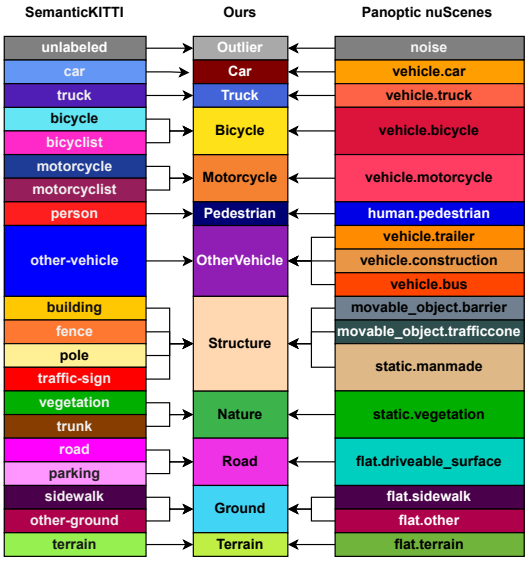

Figure 7.5.: **Joint Class Mapping of the Datasets.** The classes of both datasets used in the evaluation are remapped to match the different classes in joint categories, that are present in both datasets for a uniform class label set.

By mixing a small subset of real, annotated data of the target dataset with the generated scenes, the diversity of the dataset is increased. Furthermore the interpolation of the two domains within a single point cloud reduces the domain shift between them even further. A similar effect was noticed by the authors of another study [169], who found that merging patches of different domain sources pulls them closer together in the total distribution. The domain fusion method of this section increases this pull effect due to the structure aware fusion of the different point clouds.

7.2.6 Pseudo Labels

The technique of pulling domains together to reduce the shift between them can be applied in both semi-supervised and unsupervised ways. In the unsupervised approach, a network trained on the domain adapted data is used to create pseudo labels for unlabeled data of the target domain. The same methods as in the semi-supervised approach are then applied, using the pseudo labeled data instead of a small annotated data pool. To decrease the influence of incorrect labels, points with a class prediction confidence lower than 85% are removed. The reformulated use of the fusion method of Chapter 6.2.3 with these pseudo label has an advantage over other pseudo label approaches. Uncertain regions are not left empty, but are populated with the complete scene point clouds of the generated samples. The pseudo labels only add more information from the real point cloud, but do not remove points of the synthetic point cloud when they can not fill the empty spaces.

7.3 Evaluation

The efficacy of the lidar domain adaptation method is showcased by utilizing two publicly available datasets, namely *SemanticKITTI* [39] and *panoptic nuScenes* [26]. These datasets feature distinct lidar sensors that are mounted on vehicles of varying heights and operate in different geographic locations. Therefore, they create a significant domain gap between the different automotive lidar segmentation datasets. To facilitate the application of the domain adaptation method and enable performance comparison between the two datasets, a shared set of classes is created by remapping the classes in both datasets to a common set, as illustrated in Figure 7.5. As some previous methods [128, 42] for lidar domain adaptation use different class combinations, a direct comparison with those methods is not feasible. Therefore, only the methods proposed in [60] and [166] are used for performance comparison.

7.3.1 NuScenes to SemanticKITTI

The panoptic lidar labels, instance-wise attributes for dynamic objects, and ego-motion ground truth of the *nuScenes* dataset are utilized to remove dynamic objects from the lidar point clouds and combine all the point clouds in a sequence using their ego-motion as outlined in Section 7.2.1.

The dataset is divided into multiple sub-sequences, each containing 20 frames, acquired at a rate of 2 Hz, for a total of 10 seconds. The goal is to recreate panoptic segmentation lidar data in the structure of the *Velodyne HDL-64E* lidar sensor data. This is accomplished by summing all point clouds in each sequence and creating a three-dimensional mesh world using *Poisson surface reconstruction*. The spherical projection of the three-dimensional mesh is used to capture a depth image in the recording structure of the target sensor with a virtual orthographic camera. Minimum and maximum vertical and horizontal angles and image resolutions are defined to recreate the static scenes in the lidar structure of the *KITTI* [87] dataset. The generated data comprises panoptic labels, which offer a more comprehensive annotation of the scene. However, for the semantic segmentation evaluations that follow, only the semantic labels are utilized, as they are deemed sufficient for the specific analysis being conducted.

To evaluate the impact of each module in the domain adaptation method, an ablation study was conducted. Replacing the original *nuScenes* data with the recreated lidar frames resulted in a significant performance increase, from 19.1 *mIoU* to 30.7 *mIoU*, when evaluating the model on the validation log 08 of the *SemanticKITTI* dataset.

Further improvements were achieved by utilizing the trained network to generate pseudo labels for unlabeled data from the target sensor and combining them with the generated frames, resulting in a total *mIoU* of 34.3. Addition of the object detection cuboid labels from the original three-dimensional object detection dataset [87] as structure-aware point cloud injections, without the pseudo labels, resulted in an *mIoU* of 31.9.

Table 7.1.: ***NuScenes* to *SemanticKITTI* Ablation Study of the Presented Domain Adaption Method Using the *Cylinder3D* Network [228].** The classes are joined from the source and target dataset according to FIGURE 7.5. "GT Frames" denote the addition of a small subset of 100 annotated target frames (0.5% of the training data), while "GT Inst." is the addition of cuboid detections as point-wise labels. All *Cylinder3D* networks have been trained from scratch with the same parameters to ensure a fair evaluation. The chapter's method was compared to the unsupervised domain adaptation method of [166] and the semi-supervised domain adaptation of [60] which uses 100 annotated target frames for the training. The reported *IoUs* are listed alongside the chapter's domain adaptation methods ablation values. **Best results are shown in bold red**, *second best in italic blue text*.

	Gen. Frames	Pseudo Labels	GT Inst.	GT Frames	mIoU ↑	Car	Truck	Bicycle	Motorcycle	Pedestrian	OtherVehicle	Structure	Nature	Road	Ground	Terrain
Unsupervised					19.1	64.5	0.9	0.0	5.0	0.0	1.0	38.3	11.0	50.6	4.8	33.7
	✓				30.7	86.1	6.8	5.8	16.0	1.2	3.4	44.6	29.9	64.2	32.9	47.1
	✓	✓			34.3	88.8	3.0	1.0	16.9	0.3	1.0	49.3	42.5	74.0	51.2	49.3
Semi-Supervised	✓		✓		31.9	78.6	1.98	6.9	7.6	10.9	1.8	51.8	42.62	66.9	38.58	43.2
	✓		✓	✓	63.1	93.1	31.1	50.1	43.3	65.4	13.5	86.8	84.9	87.0	73.1	65.8
	✓	✓	✓	✓	*67.4*	*94.0*	*50.8*	*58.2*	*51.9*	**71.6**	13.9	*88.3*	*85.8*	88.2	*75.3*	67.0
Rochan et al. [166]					23.5	49.6	1.8	4.6	6.3	12.5	2.0	65.7	57.9	82.2	29.6	34.0
Corral et al. [60]				✓	46.2	87.3	27.6	29.2	26.9	34.6	*24.4*	61.7	46.4	70.3	52.3	47.4
Supervised	100 Frames			✓	49.0	91.2	1.6	8.1	2.6	30.1	6.0	83.3	85.3	*88.3*	73.3	**69.6**
	Full Target Dataset †				**75.8**	**96.5**	**84.7**	**62.3**	**53.7**	*70.2*	**53.2**	**89.5**	**86.0**	**91.0**	**79.2**	*67.3*

† The target baseline *mIoU* is higher than reported by the original authors, as the reduced joint class set as shown in FIGURE 7.5 was used, and therefore some of the bad performing classes are eliminated from the evaluation.

Incorporating 100 frames from the target domain, which constituted less than 0.5% of the original dataset, and mixing them with the synthetic data using the structure-aware point cloud fusion method, resulted in a remarkable performance increase, up to an *mIoU* of 63.1.

The final version of the semi-supervised domain adaptation method included all the previously mentioned components and pseudo labels derived from the previous network applied to unlabeled target lidar data. The final network achieved a performance of 67.4 *mIoU*, which is equivalent to 89% of the segmentation quality of the same network trained on the full target dataset which is 75.8 *mIoU*.

In comparison, training the same network directly on the 100 sampled frames of the target dataset used in the semi-supervised approach resulted in a lower *mIoU* of 49.0.

The results of the ablation study, as shown in Table 7.1, demonstrate that all the domain adaptation, injection, and fusion methods significantly improve the final segmentation quality.

When compared to two state-of-the-art lidar domain adaptation methods for semantic segmentation, the presented method demonstrates superior performance. The first method, an unsupervised approach that performs domain adaptation in the range image domain, reports an *mIoU* of 23.5 [166]. The second method, a semi-supervised approach, uses parts of the annotated target dataset and domain adaptation, resulting in an *mIoU* of 46.2 with the use of 100 annotated frames of the target dataset [60]. The method presented in this chapter achieves an *mIoU* of 67.4, while using only 100 ground truth frames, demonstrating the effectiveness of the domain fusion and injection methods in reducing the domain shift between the datasets.

7.3.2 SemanticKITTI to NuScenes

To showcase the universality of the approach, a reverse domain adaptation is carried out by using the training data from the *SemanticKITTI* dataset to replicate the content in the lidar sensor structure of the *nuScenes* panoptic segmentation dataset. The *Cylinder3D* [228] semantic segmentation network is trained on the fully unsupervised method using only generated frames and pseudo labels. A second *Cylinder3D* model is trained with the semi-supervised domain adaptation approach, using all modules from Section 7.2. Both are compared to fully supervised training on the source and target datasets, as shown in Table 7.2. An improvement in semantic segmentation quality is observed with each additional component of the method.

The performance of 7.4 *mIoU* on the *nuScenes* validation data is observed with naive training on the *SemanticKITTI* data. However, the unsupervised domain adaptation improves the performance to an *mIoU* of 29.2 which is slightly lower than the unsupervised approach by [166] with 34.5 *mIoU*. The lower performance of the unsupervised method on the *nuScenes* dataset, compared to the *SemanticKITTI* dataset, is attributed to the different vertical aperture angles of the two lidar sensors. The *VLP-32C* lidar sensor (*nuScenes*) has a larger vertical opening angle and can "see" up to ~ 40.73 m above the road surface, while the *HDL-64E* sensor (*SemanticKITTI*) is limited to ~ 3.48 m above the ground. This large discrepancy impacts the performance noticeably more for a

Table 7.2.: ***SemanticKITTI* to *NuScenes* Domain Adaption Methods Using the *Cylinder3D* Network [228].** All *Cylinder3D* networks have been trained from scratch with the same parameters to ensure a fair validation. The *IoU* of the cited papers was listed as provided by the authors. **Best results are shown in bold red**, *second best in italic blue text*.

Method	mIoU ↑	Car	Truck	Bicycle	Motorcycle	Pedestrian	OtherVehicle	Structure	Nature	Road	Ground	Terrain
No Domain Adaption	7.4	3.7	0.3	0.0	0.1	0.1	0.5	18.2	0.1	11.3	1.2	0.1
Unsupervised	29.2	72.3	0.0	0.0	0.3	0.1	4.8	59.3	38.5	77.8	25.9	42.1
Semi-supervised	*58.9*	*78.0*	*57.0*	**14.1**	**53.6**	*51.9*	**39.1**	*79.9*	*77.0*	*91.0*	52.3	53.9
Unsupervised [166]	34.5	54.4	15.8	3.0	1.9	27.7	7.6	65.7	57.9	82.2	29.6	34.0
100 Target Frames + [60]	48.3	69.0	37.7	5.5	9.4	45.4	23.5	69.0	74.7	78.8	*56.1*	**61.8**
100 Target Frames	46.3	70.3	27.1	2.0	0.1	40.3	14.7	78.1	76.0	90.7	52.1	58.0
Full Target Dataset†	**69.5**	**80.0**	**61.7**	*11.9*	*38.0*	**72.1**	*34.2*	**82.6**	**81.4**	**94.0**	**63.7**	*60.7*

† The target baseline *mIoU* is lower than reported by the original authors, as we are training from scratch.

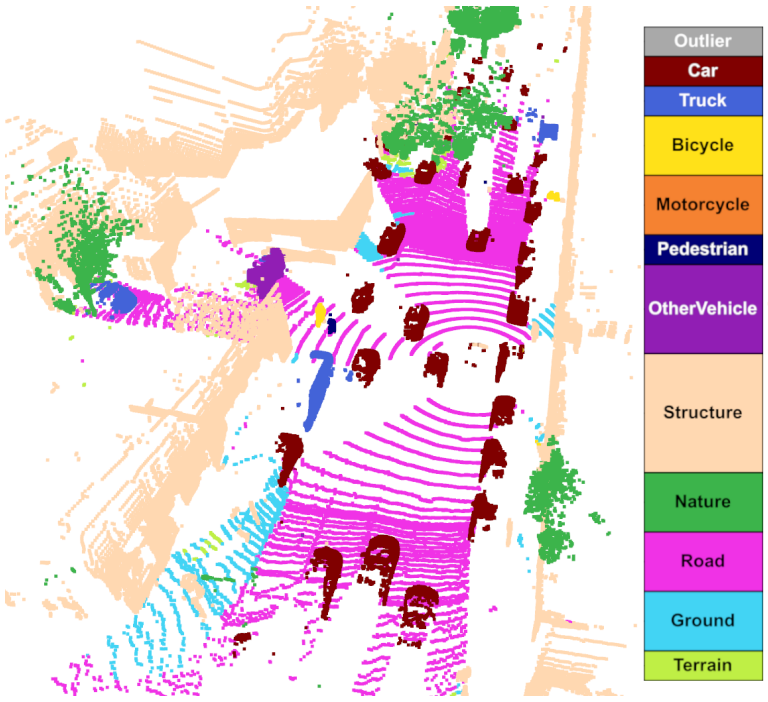

Figure 7.6.: **Inference Results of the *Cylinder3D* [228] Semantic Segmentation Network Trained on *NuScenes* Data Recreated in the Structure of the *Velodyne Alpha Prime* Sensor.** The network performs well for most classes even at distances > 50 m, but it exhibits a high uncertainty in ambiguous regions, especially vegetation areas are prone for false detections of various classes.

three-dimensional point-wise domain adaptation than a range image variant, as the latter samples the closest point in the image to fill the gaps [128, 166].

The best performing semi-supervised method utilizes 0.36% of the original target training data, and reaches a final *mIoU* of 58.9, as shown in Table 7.2. To prevent data leakage, injection instances are sampled from the same 100 frames. This results in a performance of 85% when compared to a network trained on the fully labeled target dataset which has an *mIoU* of 69.5. The presented semi-supervised method even outperforms the fully supervised network on three out of 11 classes.

Compared to the state-of-the-art semi-supervised domain adaptation method by [60], which reaches 48.3 *mIoU*, the presented semi-supervised approach shows a significantly higher performance with an *mIoU* of 58.9. Even their method with 500 labeled frames does not reach a comparable performance with it's *mIoU* of 52.3.

7.3.3 NuScenes to Velodyne Alpha Prime

In this section, the lidar domain adaptation method was applied once again to the *nuScenes* dataset. Synthetic data in the structure of the high-resolution lidar sensor *Velodyne Alpha Prime* was generated. Furthermore raw automotive lidar data was captured in multiple scenarios in Wuppertal, Germany to get a pool of unlabeled real data for the pseudo labels.

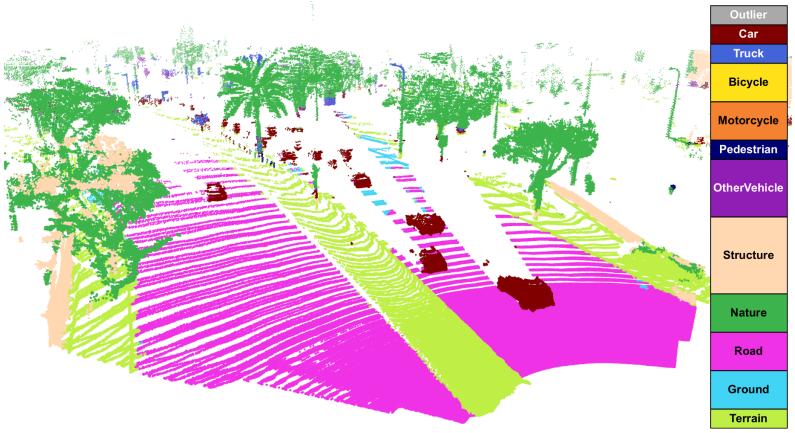

Figure 7.7.: **Inference Results of the *SalsaNext* [61] Semantic Segmentation Network Trained on *SemanticKITTI* Data Recreated in the Structure of the *InnovizTwo* Sensor.** Due to the low number of cuboid instances for our injection module, the absence of segmentation ground truth and the very large vertical field of view, the performance on this sensor is not as precise as on the *Velodyne Alpha Prime* and even less than the two open source datasets.

The target lidar has a vertical resolution of 128 non-uniform lidar channels, four times the resolution of the *nuScenes* lidar, and a horizontal resolution of 1800 points per scan line, resulting in twice the horizontal resolution of the *nuScenes* lidar data. Additionally, the range of the target sensor is 100 m farther with 300 m.

Similar to the previous section, all points within a scene were aggregated to gather as many original lidar measurements as possible, resulting in a comparably sparse point cloud due to the lower resolution of the source lidar. To address this, a meshing process was employed to connect the point cloud and cover the entire visible surrounding.

Two off-the-shelf three-dimensional bounding box algorithms [127, 178] were applied to unlabeled target data of the *Velodyne Alpha Prime*, and a *Kalman filter* [115] was used to filter out most false detections.

The approach discussed in Section 7.2.4 was employed to extract lidar points from the detected cuboids and integrate them into the produced training dataset as semantic instances. Figure 7.6 presents the qualitative outcomes of the trained semantic segmentation model. However, due to the absence of any publicly accessible semantic or panoptic segmentation dataset for the *Velodyne Alpha Prime* sensor, a quantitative assessment cannot be provided.

7.3.4 SemanticKITTI to InnovizTwo

To showcase the domain adaptation capability of the method, it was applied to another dataset with a different lidar sensor that lacked segmentation labels. The *InnovizTwo* lidar sensor was selected, which has a high resolution, directional configuration with a limited aperture angle of $120° \times 40°$ and can sense objects up to 300 m away. The *InnovizTwo* has a much higher point density in the given direction than the *Velodyne Alpha Prime*. The data from the *InnovizTwo* sensor was obtained

from a self-supervised object detection challenge, and the domain adaptation was performed from the low-resolution, 360° rotating lidar sensor of the *SemanticKITTI* dataset to the high-resolution directional *InnovizTwo* sensor.

Point-wise instances were defined for the semi-supervised injection module of the domain adaptation using the provided cuboid labels of 100 annotated frames. The results of the trained *SalsaNext* semantic segmentation model for the *InnovizTwo* data can be seen in Figure 7.7.

Similar to the previous section, since there is no annotated dataset available for the *InnovizTwo* lidar sensor, a quantitative analysis cannot be provided.

7.4 Conclusion

In this chapter, a novel domain adaptation method to recreate annotated segmentation lidar data in the structure of different lidar sensors was presented.

The evaluation demonstrated, that the proposed method improves semantic segmentation via domain adaptation by up to +21.2 *mIoU* compared to the current State of the Art. An extensive ablation study was conducted to show the influence of each module in reducing the domain gap between generated and real data.

The method operates solely at the data level and can be used with any lidar semantic segmentation model. This is especially useful for future uses, as the state of the art for segmentation models is a constantly changing and improving area of research.

In the future, the application of the method to panoptic segmentation networks and three-dimensional bounding box detectors represents a good opportunity for further development and improvement of the method.

Part III

Expanding the Horizons of Autonomous Driving: Novel Applications of Lidar Segmentation

8 Driving Forward with Lidar Segmentation:

Innovative Applications in the Automotive Industry

As demonstrated in previous chapters, lidar segmentation is a powerful tool that enhances the utility of point clouds. By dividing the data into distinct segments, it becomes more feasible for human observers to understand the structure of the sensor's environment. However, the true value of segmented point clouds lies in their applications for driver assistance systems and autonomous vehicles. In this chapter, several applications of segmented point clouds will be presented. These not only demonstrate the value of segmentation algorithms but also explain the necessity of segmenting lidar data to support such applications.

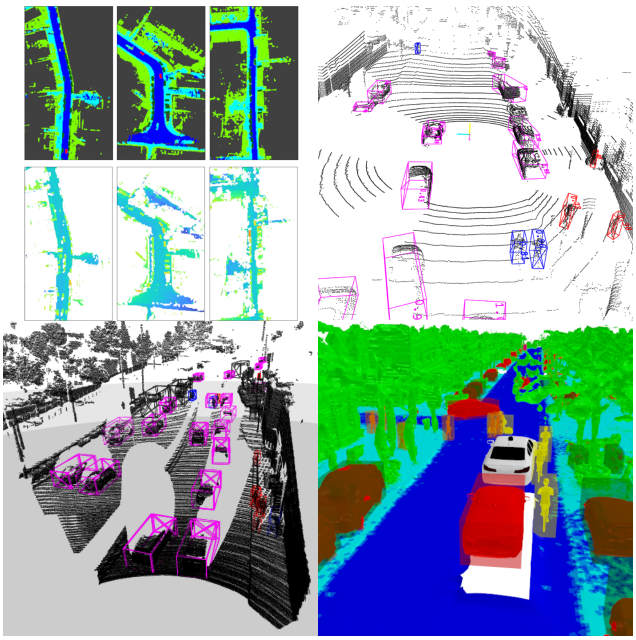

Figure 8.1.: **Applications of Lidar Segmentation.** Examples of the new applications of lidar segmentation presented in this chapter.

Segmenting lidar data by class and instance provides additional information that can be useful for decision-making. For example, a lamppost is not as important to an autonomous vehicle as a pedestrian. The latter can change its position in the future. This distinction is not possible from unordered raw lidar data, but can be facilitated by semantic segmentation. By classifying the static background, not only the dynamic pedestrian can be clearly separated from the rest of the lidar points, but the background itself can also be divided into different spatial regions. For example, the spatial segmentation of road surfaces versus that of a curbstone is important for trajectory planing. A cuboid detection of a pedestrian merely indicates its presence. A semantic and panoptic segmentation can provide additional information, such as the fact that the pedestrian should not be avoided via the sidewalk, but that the free road surface should be used instead.

Lidar segmentation provides a comprehensive representation of the environment. It enables well-informed decisions based on a holistic understanding of the surroundings. This is an improvement over reactive decision-making based on limited detections, and can improve the safety and efficiency of autonomous systems.

Currently, the direct processing of three-dimensional segmented point clouds is computationally too expensive for online decision-making algorithms. In addition, segmentation networks have a certain latency before the processed point cloud can be passed on to the decision-making algorithm. As a result, segmented lidar point clouds are rarely used in live applications. However, their high information content makes them suitable for providing a ground truth for offline training of other sensors and algorithms and as a tool for human evaluation and refinement of a given scene.

In the following sections, four novel methods are presented, each in their own dedicated section. All methods have lidar segmentation as a common basis. The first method entails the reformulation of lidar segmentation data to create semantic grid maps which are used to train online segmentation algorithms based on radar data. The second method introduces a novel CPU-based real-time lidar detection algorithm which is based on panoptic lidar segmentation. The third method describes a simulation environment for Advanced Driver Assistance Systems (ADAS) re-simulation which is based on lidar segmentation data recorded in the real world. The last method demonstrates the use of lidar segmentation training augmentation and domain adaptation techniques to train a lidar object detector on a limited dataset.

The following main contributions are provided in this chapter:

- A novel semantic grid map generation algorithm.
- A novel online CPU-based object detection algorithm.
- A novel closed loop re-simulation system.
- Novel augmentation and adaption methods to train semi-supervised object detection networks

8.1 Lidar Segmentation for Radar Segmentation

To extract relevant information from lidar segmentations for subsequent applications, it is typically necessary to reformulate the underlying point cloud segmentations. Occupancy and semantic grid maps are common types of representations used in robotics and autonomous systems to represent the environment in a structured and simplified way.

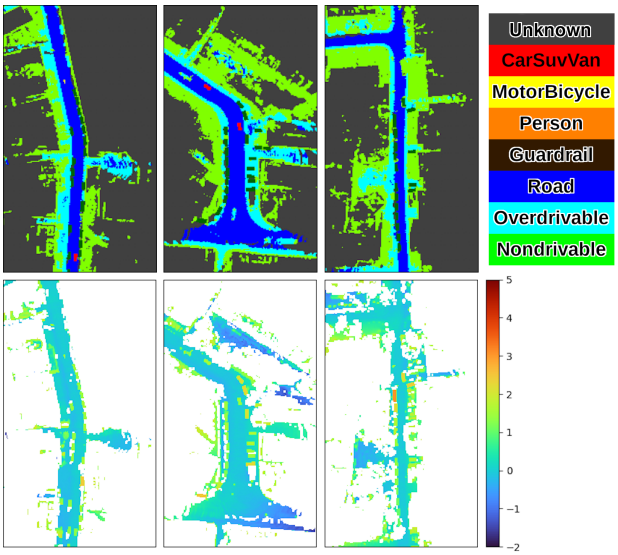

Figure 8.2.: **Lidar Segmentation Grid Maps.** Bird's eye view maps of the environment around a vehicle provide structured and sensor-agnostic information that can be used for decision algorithms and path planing. The top row shows the semantic information in a 50 cm by 50 cm resolution 80 m to the front, 60 m to the back and 40 m to the sides of the vehicle. The bottom row shows the associated height information for the same grid cells.

Lidar sensors which use lasers to measure the distance to objects in a 360° field of view, can be used to create semantic maps from point clouds [44, 80]. Yet radar sensors are a more cost-effective method to create semantic maps [143, 160], especially for consumer applications in the automotive industry. These are much less expensive than lidar sensors and provide a comparable depth measurement of the environment. However, due to the complexity of radar sensor data, preprocessing is required to obtain the desired information. Also automotive radar sensors have a significantly lower resolution than lidar sensors, meaning they are not as accurate in detecting the shape and size of objects.

This section presents an automatic ground truth generation algorithm that takes advantage of the strengths of lidar sensors over radar sensors. The algorithm is based on the lidar segmentation process, to generate high-resolution semantic grid maps that can be utilized as ground truth for radar sensors. By employing this method, it becomes possible to develop and train grid map segmentation networks that can produce structured two-dimensional semantic grid maps from complex radar data.

The method presented in this section combines several state-of-the-art algorithms from the fields of semantic lidar segmentation, three-dimensional bounding box detection, non-causal tracking and *Simultaneous Localization And Mapping* (*SLAM*) to create high resolution and far-range pseudo ground truth semantic grid maps for the training of radar networks.

8.1.1 Related Work

Occupancy grid maps are a type of representation used in robotics and autonomous systems to represent the environment. They are created by discretizing the environment into a grid of equally-sized cells. Each cell is then marked as either occupied or unoccupied based on the presence or absence of objects in that region [75]. This allows a robot to build a high-level understanding of its surroundings and can be used for a variety of tasks such as localization, navigation, and path planning [46, 209, 174]. Occupancy grid maps can be created using various types of sensor data, including lidar, camera, and radar.

Occupancy grid maps are often used in conjunction with other types of representations, such as semantic maps or height maps, to provide a more detailed and accurate understanding of the environment [79, 173].

Initially, semantic extensions to these occupancy maps were limited to information that explicitly predicted the road surface [130] or combined free-space and obstacle height estimations [38]. Further extensions of these methods assign a specific class to each pixel of the grid map and combine the semantic information with occupancy information. These are used to build a two-dimensional bird's eye view map of the entire environment [173].

Semantic grid maps can be created from different sensor sources. Camera sensors were used, because of their high pixel density and the low price. However, due to the lack of depth information, no certainty is guaranteed and several additional components were needed: for example, a stereo camera setup [130], a sensor fusion with additional depth information [139] or a neural network estimation of the three-dimensional world extended from the monocular image [136].

Other approaches based on lidar data use deep learning methods, to predict dense two-dimensional grid maps from single lidar point clouds [44, 80, 157]. These approaches either introduce uncertain labels to unknown and occluded regions due to the network's guesses, or are artificially masked to the area visible to the sensor at a given time.

An approach that was used as a basis of the method presented in the next section, creates semantic grid maps from lidar data that were annotated by human labelers. The point clouds are rasterized into a discrete grid pattern, and assigns the most frequent label of each bin. This produces a very sparse and coarse grid map, since the bins have to be large in order to have at least one semantic labeled point in most bins. Furthermore, the resulting sparse grid maps are limited to what the sensor can see at a given time. Occlusions and moving objects can therefore create large uncertain regions.

An iteration of this approach involves using a Simultaneous Localization and Mapping (SLAM) method to capture the movement of the ego vehicle. The trajectory is used to overlay sequential lidar frames that had been annotated by humans. This increases the number of lidar points and thus fills some of the gaps that single lidar frames are not able to capture. These approaches are not scalable, as they require manual annotations and the sequential point clouds have to cover the same area in order to bolster the points of a given area. However, highly dynamic environments such as highways have comparably few point clouds that map the same region before the car leaves it again.

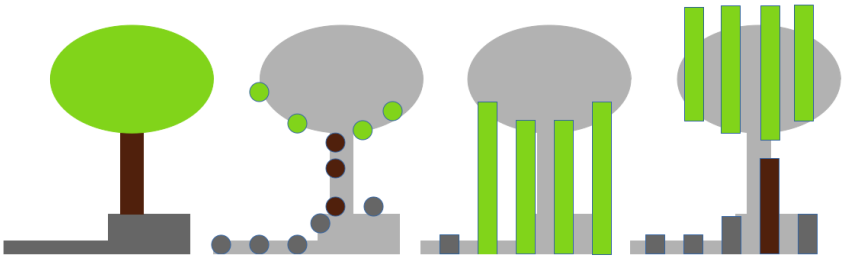

Figure 8.3.: **Comparing Information Representation: Real World, Semantic Segmented Lidar Frame, Two-Dimensional Grid Map, and Three-Layer Grid Map.** The two-dimensional grid map classifies the entire space under the tree canopy as occupied, while the three-layer grid map reveals a clear driving corridor extending to the curb stone.

The method presented in the next section combines the use of sequential lidar data, ego-motion estimation, three-dimensional bounding box detection, lidar segmentation and three-dimensional surface estimation methods to create high resolution, far-range birds eye view maps of the entire surrounding for the training of radar sensors with a very high certainty.

8.1.2 Method

The method described in this section creates high-resolution birds eye view maps. These grid maps encode the three-dimensional and semantic lidar information of the environment as a three-layer grid map. The first layer encodes the semantic label information of each pixel. The second layer holds the information of the tallest point in each cell, excluding overhanging objects. The third and last layer also provides a height information, but in a worms eye view map, i.e., seen from below. This third map provides the lowest, overhanging point of each cell. The two geometric values together provide the information of the drivable corridor for a given cell. These maps are sensor independent and can be used to train, for example, semantic segmentation algorithms for radar data. A schematic comparison is presented in Figure 8.3, illustrating the information representation of the real world, a semantic segmented lidar frame, a two-dimensional grid map, and the proposed three-layer grid map.

To create this efficient grid map data, three independent algorithms that process the raw lidar data are combined.

The first method is an automatic generation of three-dimensional bounding box annotations. For this, an ensemble of several existing detection networks [127, 178] is applied to the lidar data and combined by a common Kalman tracker [115] to produce reliable automatic bounding box labels from the detections.

The second method is an automatic semantic segmentation of the lidar point clouds. Each individual lidar point is assigned a specific class. Unlike the previous method, there is no combination of multiple algorithms, and consists of a single *Cylinder3D* [228] network trained with the augmentation methods of Chapter 6.

The third method estimates the ego motion. It is a novel combination of a lidar *Simultaneous Localization and Mapping* (*SLAM*) method [177] with recorded information of the host data, such as speed and yaw rate.

The first method is not a new contribution of this dissertation. The interested reader is therefore referred to the corresponding sources for more detailed information on the two detection networks [127, 178] and the Kalman tracker [115].

The second method, lidar semantic segmentation, has already been presented multiple times throughout this thesis. The chosen network *Cylinder3D* was presented in Section 5.3.1. The training and evaluation of the model can be seen in Section 6.3.1.

The third method is a novel combination that was developed in the course of this thesis and published as a patent application [100].

Reliable and Precise Ego Motion Estimation

Accurate ego motion estimation is a vital aspect in generating a reliable map. To achieve this, a combination of SLAM and host vehicle information has been developed for the specific application at hand. The ego motion estimation method is briefly presented below:

Dead Reckoning is a technique for calculating the value of a time-dependent variable by adding changes to a previous value. This technique provides a rough approximation of the vehicle's motion by using it's previous position and incorporating its host data, i.e., it's speed and turn rate [78]. However, due to the accumulation of small errors the predicted motion quickly drifts, resulting in an offset of several meters from reality after just a few seconds [49].

The ego motion obtained from a *SLAM* algorithm is typically highly accurate. However, any error in the estimation can result in a catastrophic failure. For instance, when navigating through environments lacking clear features, such as on a freeway, other vehicles moving parallel to the ego vehicle may be incorrectly interpreted as stationary objects, leading to the estimation of self-motion at a significantly slower pace, such as walking speed, instead of the actual highway speed.

To overcome this challenge, a novel approach was developed in this thesis, which involves a combination of the two methods. The approach involves matching the accurate ego motion estimates obtained from the *SLAM* algorithm with the relative motion data obtained from the host vehicle at regular intervals of one second. If the error between the two trajectories is small, it can be assumed with high confidence that the *SLAM* algorithm has produced a more precise ego motion estimate due to its high precision. However, if the error between both methods exceeds a predefined threshold, it is likely that the *SLAM* algorithm has produced an erroneous estimate, and the interval is filled with the dead reckoning of the vehicle data. This method ensures that a seamless sequential ego motion estimate is created, which combines the high precision of the *SLAM* algorithm with the high accuracy of the vehicle data. This results in an ego motion estimate that is both precise and accurate, ensuring a valid ego motion estimation.

In Figure 8.4 ego motion outputs are shown for two scenes, one on a highway and another in an inner city scene, showcasing the issues of *SLAM*-only and host-only methods compared to

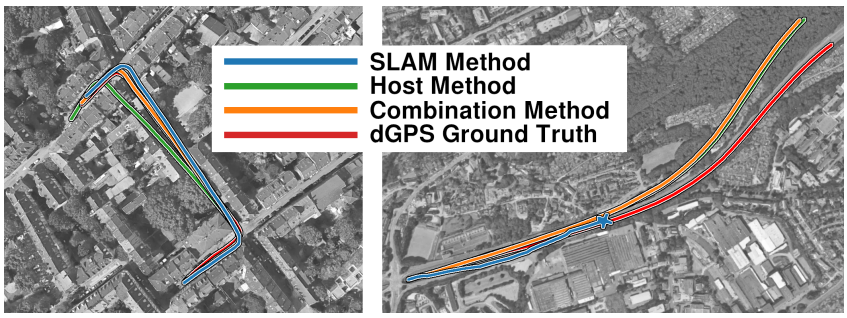

Figure 8.4.: Ego Motion Combination Method Compared to *SLAM* and Host Vehicle Based Ego Motion. The *SLAM* ego motion (blue) is generally more precise than the host vehicle dead reckoning (green), but can fail entirely in scenes without meaningful structures of the static background. Especially highways offer few significant objects for *SLAM* algorithms. *Left:* The presented combination method (orange) achieves a significant improvement of the final ego motion, that is very close to the true measured *Differential Global Positioning System* (*dGPS*) path that the target vehicle drove (red). *Right:* The presented method recognizes the errors in the *SLAM* and uses *Host* information.

the presented combination approach. The ground truth was measured using a *Differential Global Positioning System* (*dGPS*). *DGPS* systems are much more precise and reliable than conventional *Global Positioning Systems* (*GPS*) systems. A mobile *dGPS* receiver can significantly improve the accuracy of its satellite-derived *GPS* position, from around 20 m down to centimeter accuracy, by using short-range signals from ground-based transmitters [76]. Unfortunately *dGPS* are very costly and therefore not available beyond research and development vehicles.

The performance of the *SLAM* method on the inner city log was found to be excellent, with only a slight deviation from the ground truth *dGPS* data. However, the motion estimated from the *Host* data begins to drift after the first turn. The presented combination method bridges a short failure of the *SLAM* with the *Host* data for an even better performance than any one of the two.

In contrast, the highway scene highlights the potential risks of relying solely on the *SLAM* algorithm. In an environment lacking distinct features, the *SLAM* algorithm often produces an erroneous ego motion estimate. However, the combination method detects the significant difference between the two ego motion estimates and switches to using the *Host* motion data for most of the log. Although the combination method cannot correct the drifting motion of the *Host* data, it ensures that the ego motion estimate is at worst imprecise but not entirely incorrect.

Combination of Independent Algorithms

The presented novel method further processes the three independent algorithms, namely the semantic segmentation, bounding box automatic label generation, and ego motion estimation.

Firstly, the point cloud is analyzed to differentiate between static and dynamic elements, including houses, road surfaces, poles, parked cars, pedestrians, moving cars, bicycles, and trucks. This is achieved through the use of semantic segmentation predictions and automatically generated bounding boxes throughout the recorded time interval.

To separate the dynamic elements from the static ones, the Kalman tracker of the bounding boxes assigns a velocity attribute to each tracked automatic bounding box label. As a result, dynamic cars can be distinguished from static ones through this process.

The dynamic points are then extracted and removed from the point cloud, leaving only the static points. This allows for the use of the combination ego motion method described in Section 8.1.2 to create a denser point cloud representation of the environment. Despite the increased point density, many areas remain empty due to the sparsity of lidar data at larger distances.

To improve coverage of the static environment, all points in the condensed cloud are used to create a three-dimensional mesh object using the *Poisson surface reconstruction algorithm* [116]. The vertices of the mesh object are compared to the original scene point cloud and a k-nearest-neighbors algorithm is applied to select the ten closest points in the original cloud for each vertex. This is done to match the semantic segmentation labels of the original points to the vertices. Each point's label is determined by its corresponding class frequency, with certain classes being given more weight as they are deemed more critical. For instance, the "Guardrail" class is more crucial than the "Road" class because ignoring the former could result in an accident.

The process is described in more detail in Chapter 7.2.2 for the application of domain adaption. The steps are visualized in Figure 7.2 (a - c) on page 98.

For dynamic objects, the three-dimensional box labels from the automatic bounding box annotation method are used to transfer these boxes to the three-dimensional mesh world at the appropriate times and locations. By combining the static mesh world with the dynamic boxes, the complete spatial and temporal course of the recorded time period can be displayed.

To use this holistic world representation for training radar segmentation networks, the data is reduced to a pseudo map representation. Methods from the computer graphics domain are used for this step. Two virtual cameras take "pictures" from above and below the target vehicle to create a semantic map, as well as two $2.5D$ elevation maps. The first camera, looking from "top to bottom," records the height of the tallest object around the vehicle in every defined grid cell. The second camera, capturing an image of the mesh world from below the vehicle, records the lowest protruding object from above, down to the car. The mesh structure of the world enables this simple formulation, as the face elements of the meshes have clearly defined surface normals. Faces seen from the correct side are visible to the virtual cameras, while the backside of surfaces is invisible to them as shown in Figure 8.5. Therefore, the virtual camera below the ego vehicle, does not capture the ground or vehicles, but only faces that are above the vehicle, with the faces facing towards the bottom. In the same way, the virtual camera above the ego vehicle is not blocked by overhanging objects, as the faces are oriented towards the vehicle. In this way, regions such as bridges, trees, signs on highways, and tunnels can be mapped with their height in a data format that emulates the two-dimensional nature of radar segmentation maps. They still encode the three-dimensional information of the environment for the most important objects of each grid cell, i.e., the information closest to the vehicle.

These three maps, the semantic, the bird's eye view height map and the worm's eye view height map, create a pseudo three-dimensional representation of the environment with a much more efficient data structure. Using this data, radar segmentations can be trained, for the classes of the lidar

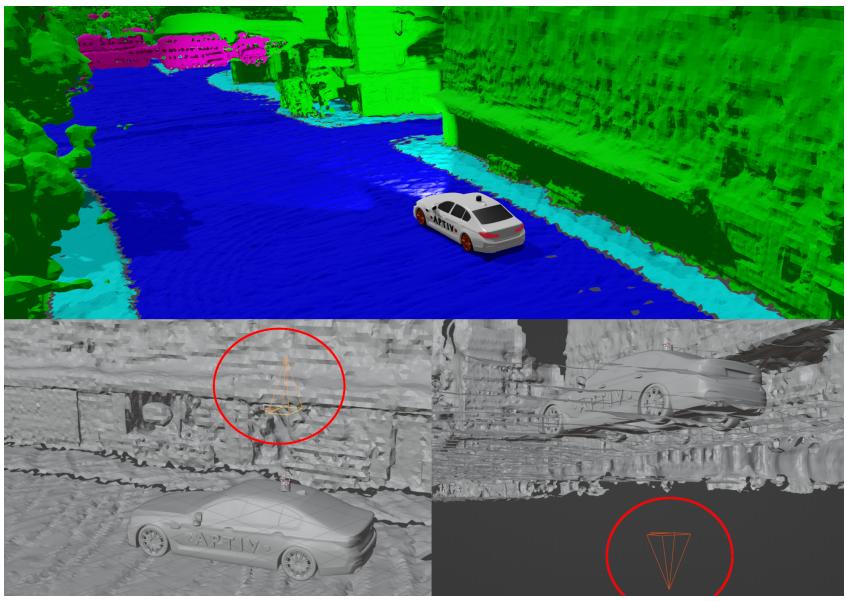

Figure 8.5.: **Mesh World Rendering.** The environment mesh (*top*) is captured by two virtual depth cameras. These cameras are located above (*bottom left*) and below (*bottom right*) the ego vehicle.

segmentation as well as the height and the overhang height for live applications using only radar data.

The presented method can be scaled to high resolution maps due to the underlying world representations as mesh objects and can be enlarged to cover high range applications. The method is not limited to the range and point density of the lidar sensor but can be used beyond the lidar capabilities due to combination of the segmentation, the cuboid detection and the ego motion estimation.

8.1.3 Results

At the time of writing, there is no available ground truth dataset to quantitatively evaluate the proposed method for semantic and height map generation. There are three dimensional voxel elements of the *SemanticKITTI* dataset for a scene completion task [39], that is similar to the task at hand. The creators encoded three-dimensional information in a Cartesian voxel grid and assigned semantic labels. The data format suffers immensely from a missing exclusion of dynamic objects due to which the ground truth is unreliable. The *nuScenes* dataset has high definition maps for the recorded scenes of their logs, but this map offers only the semantic information about the static environment. Furthermore the *nuScenes* lidar sensor is artificially reduced to a 2 Hz frequency which greatly decreases the capabilities of the proposed method.

The evaluation of the presented method was therefore purely qualitative. In Figure 8.2 multiple scenes of the proposed method are shown. The resulting maps can be tuned to specific user settings depending on the desired range and resolution. Figure 8.6 illustrates a comparison between the original lidar point cloud and the reconstructed scene using the final $2.5D$ grid maps.

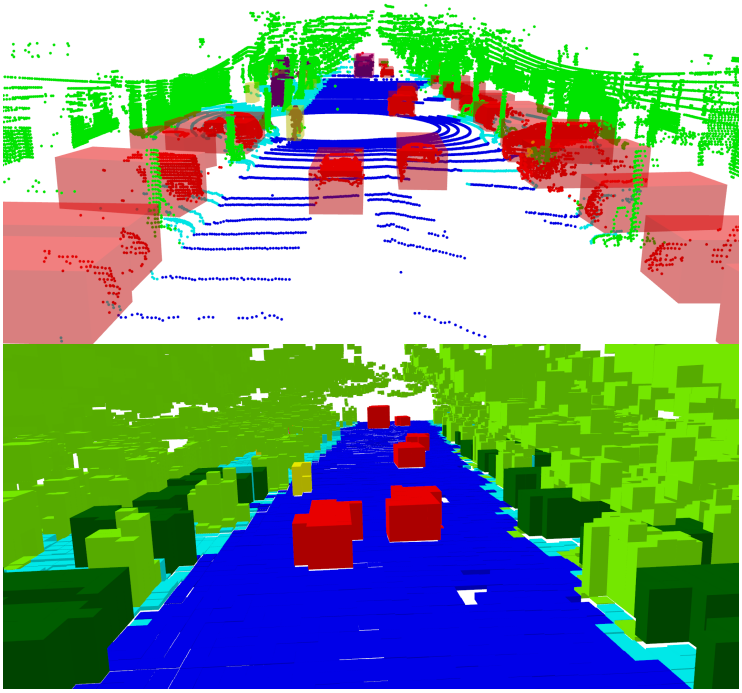

Figure 8.6.: **Three-Dimensional Reconstruction of the Encoded Grid Map Data.** The three-dimensional lidar point cloud (top) shows the target scene as recorded by the sensor with the semantic segmentation and cuboid detection labels. The method outlined in Section 8.1.2 creates an efficient multi-layer grid map format, encoding the most important spatial and semantic information of the scene to recreate the full scene (bottom).

The resulting multi-layer grid maps encode the semantic label and the important aspects of a valid driving corridor, with three values per grid cell. This data format is highly efficient, as it represents three-dimensional information in two two-dimensional grid images. Moreover, it allows for easy comparison to other modalities that produce grid maps, such as camera-based or radar-based online algorithms.

8.1.4 Conclusion

In this section a novel method for semantic, height and overhang-height encoded grid maps was presented. The maps represent a sensor agnostic segmentation of the environment around a vehicle. They are created by a combination of machine learning, *SLAM* and other heuristic algorithms to generate accurate and detailed maps of the environment. These can be used for the training of segmentation algorithm used for autonomous driving and robotic navigation systems.

The method shows generally very promising results over a large variety of scenarios, ranging from inner city to highway areas. A few issues were identified during the visual inspection of the results, namely missing detections in both the cuboid as well as the segmentation labels, which can lead to false static objects in the middle of the road. As a future task, these issues are to be addressed via moving object segmentation, and semantic segmentation with temporal consistency as future work.

Overall the presented method is effective in accurately representing the true environment and presenting meaningful information in a compact data structure. The method represents a valuable contribution to the field of environment segmentation and mapping, as it creates a new data structure for novel algorithms, by combining present independent algorithms. Combining multiple independent algorithms increases the overall reliability of the method and utilizes the advantages of each component while providing an additional fail-safe mechanism. By combining the *SLAM* with the *Host* motion and integrating segmentation and cuboid labels, the method is capable of filtering out errors in each other. These features make it a useful tool for various applications, including trajectory prediction, path planning, and map-less autonomous driving.

8.2 Lidar Segmentation for Online Detection

The methods presented in Chapters 3 and 5.2 were not originally designed for instance and panoptic segmentation application. The goal was to develop a lidar object detection method that can run live in a vehicle while using as few resources as possible for a publicly funded project.[1]

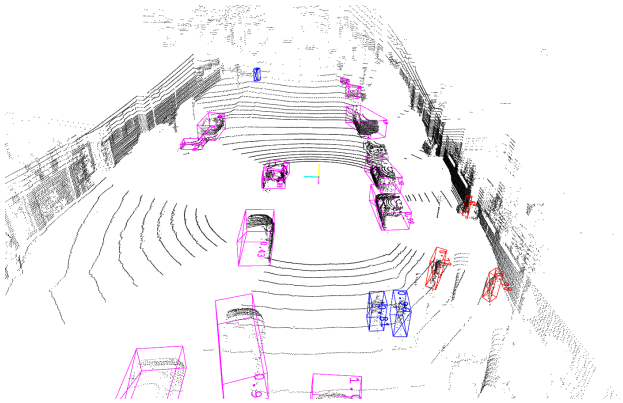

Figure 8.7.: **Online Object Detection From Panoptic Segmentation.** The bounding boxes are created with the clustering method of Chapter 3 together with the lidar cluster classification method outlined in Section 5.2. The purple boxes represent car detections, the red boxes are pedestrian detections and the blue boxes are bike detections.

The detection of other road users, especially vulnerable ones, was designed into an overall system for interaction with them. This means that after the recognition of all road users in the environment, further subsystems followed [92]. This included tracking the road users [172], intention recognition [202], prediction of future movement [172], gesture [210, 48] and facial expression recognition [114], and finally the decision algorithms for the ego vehicle itself [102].

State of the Art object detection methods at the time [230, 24, 53, 216, 127] performed very well for the detection of the other road users, but most methods were not capable of real time applications, i.e., an inference at the sensor's frequency. Furthermore, all of these required an entire GPU solely for the detection of other road users, thus blocking the resources for the subsequent modules.

The method described in this section is different from previous work because it was designed to run in real time on a single CPU core, thus it is much more efficient and left more than enough time and resources for the following algorithms. The proposed method uses a combination of clustering and image classification, to identify objects in the point cloud and yields three-dimensional bounding boxes as output for the following processes. It starts by dividing the point cloud into smaller segments, clustering points that belong to the same object. Then, it uses a classification algorithm to identify the type of object in each cluster. Finally, it fits a cuboid around the classified object cluster to create an object detection cuboid label. This approach allows the algorithm to run quickly and accurately on a single CPU core, making it a valuable tool for applications that require real-time object detection on lidar point clouds.

[1] This work was a result of the research project @CITY Automated Cars and Intelligent Traffic in the City: Subproject 7 - Interaction with weaker road users

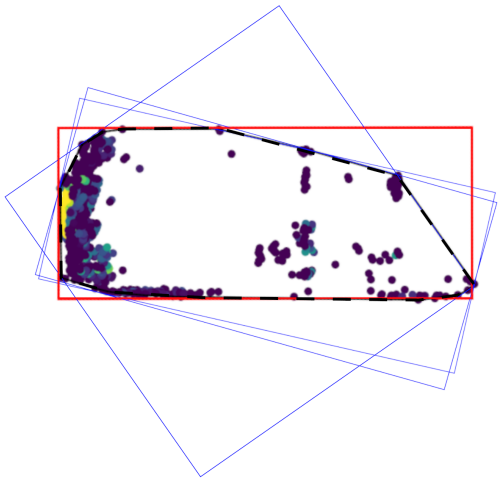

Figure 8.8.: **Bounding Box Size Estimation via a Convex Hull.** Each classified instance is projected onto the x,y plane. A convex hull (dashed black polygon) is applied to the point set to calculate the smallest rectangle (red) around the shape out of all possible rectangles (blue).

8.2.1 Method

In the CHAPTERS 3 and 5.2, the real-time capability of the panoptic segmentation method used here was presented in detail. However, panoptic segmentation is not the best data structure for the subsequent algorithms in the system. Panoptic segmented lidar data is still multidimensional with millions of points that are captured per second. This data density is too large for further processing by a real time capable system. For this reason, the online capable panoptic segmentation is reformulated to a real-time capable object detection which works without the use of a GPU.

The classified clusters for each of the defined classes *"Car"*, *"Truck"*, *"Pedestrian"* and *"Bike"* are reformulated to instance bounding boxes. This step simplifies the data structure for the following parts of the full vehicle system. An example of the resulting bounding boxes are shown in FIGURE 8.7.

The box dimensions and orientation are defined by a *minimum area rectangle* algorithm. As a first step the minimum and maximum value of the Cartesian z dimension (height) are extracted. Next, for an efficient implementation, the point cloud is collapsed along the z dimension in order to turn the point cloud into a two dimensional point set. A convex hull [167] in two dimensions is formulated around the point set as shown in FIGURE 8.8. This is done, as the orientation of the *minimum area rectangle* has to be the same as one of the edges of the convex hull of the point set. For this reason, the angle to the original orientation in the $x-y$ plane is calculated using the *arctan* for each edge e between adjacent vertices of the convex hull

$$\phi = \arctan(e) \ \% \ \left(\frac{\pi}{2}\right), \tag{8.1}$$

where % is the modulo operator to normalize ϕ to the range $\left\{\mathbb{R} \mid 0 < x < \frac{\pi}{2}\right\}$.

The process of calculating ϕ is illustrated in FIGURE 8.9.

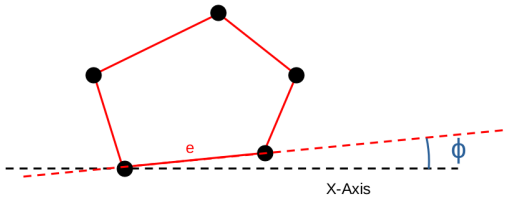

Figure 8.9.: **Points with Convex Hull and Edge Angle Orientation.** Each edge of the convex hull is indicated by a line segment. The angle between a given edge e and the x-axis is computed using the arctan function.

For each edge angle, the vertices of the convex hull are rotated, and the minimum and maximum values of the x and y dimensions are determined by a vectorised minimum operation in an efficient manner. Since the target area is a rectangle, the *arctan* is limited to a 90° angle. The resulting rectangle areas are calculated and the minimum area for each cluster is determined. The corner positions of the minimum rectangle, together with the previously determined minimum and maximum height, results in a three-dimensional cuboid that contains the position, dimensions and rotation of the corresponding instance, while the class was already predicted by the lidar image classification outlined in Chapter 5.2.1. These cuboids are better suited for real-time applications, as subsequent algorithms only need to process this vector abstraction of the objects which consists of only eight values instead of a long list of points associated with the given object.

8.2.2 Evaluation

There are certain disadvantages of this method to the previously mentioned state-of-the-art algorithms [230, 24, 53, 216, 127] for object detection in lidar data. The dimensions of the cuboids depend on the lidar points themselves. In case of partial occlusions the cuboid will only reflect the visible part of the instance, as shown in Figure 8.7, the car at the far left of the origin. Also, the orientation is not related to the orientation of the road user, but can point in any direction.

Despite these restrictions, the presented method achieves a competitive recall in real time on a single CPU as shown in the following.

Recall

The presented reformulation of the panoptic segmentation to an object detection task was evaluated on the *SemanticKITTI* [39] dataset instead of a three-dimensional bounding box dataset such as the original *KITTI* dataset [86]. This was done as the cuboid orientation and size diversion did not result from the underlying detection algorithm but the following *minimum area rectangle* fitting.

As a metric for evaluating the detection quality, the *Average Recall* was used. For each ground truth instance, the classified cluster with the highest *IoU* score was compared. A ground truth instance was considered "found" if the *IoU* score exceeded a defined threshold and the class prediction was correct. Thus, the evaluation was still linked to the point-wise segmentation quality.

Table 8.1.: **Average Recall of the Detection Method.** The *Recall* for the three *IoU* thresholds 0.5, 0.75 and 0.95 as well as the *Average Recall* are reported once for the clustered instances of [99] and once for the ground truth instances as input for the classification approach of this section.

Method	$R_\mu \uparrow$	$R_{0.5} \uparrow$	$R_{0.75} \uparrow$	$R_{0.95} \uparrow$
FLIC Instances [99]	40.7	44.1	41.9	31.4
Ground Truth Instances	55.4	-	-	-

The *Recall* (R) for a given threshold is defined as

$$R_{thr} = \frac{1}{N}\sum_{i=1}^{N} a_i, \text{ with } a_i = \begin{cases} 1, & \text{if } \max_{j=1}^{M} IoU(N_i, M_j) \geq IoU_{thr} \\ 0, & \text{otherwise,} \end{cases} \tag{8.2}$$

where N is the number of instances, M is the number of clusters and R_{thr} is the *Recall* above the IoU_{thr} threshold. For the *Average Recall*, the proportion of ground truth instances that were found was calculated for ten *IoU* bins from 0.5 to 0.95 in steps of 0.05 and averaged over all ten bins. Table 8.1 shows the *Average Recall* (R_μ) defined in this way for the ten bins as well as the *Recall* of the three selected bins 0.5, 0.75 and 0.95. The detection was also validated for the influence of the selected clustering method by extracting the instances directly from the ground truth and predicting the class label via the lidar image classification. The result has either an *IoU* of 0 or 1 and is therefore only shown for the *Average Recall* in Table 8.1.

In retrospect, this evaluation has two major weaknesses: First, the *IoU* is calculated directly from the points and not from the overlap of two-dimensional or three-dimensional cuboids, as common in object detection. Thus this method is not comparable with state-of-the-art methods [230, 24, 53, 216, 127]. Second, false positives are not included in the *Recall* metric, because only the prediction results overlapping the ground truth instances are evaluated.

The comparison of the results shown in Table 8.1 to the class-agnostic evaluation of the *FLIC* in Chapter 3.3 presents additional insights. The classification *CNN* reduces the *Average Recall* to 40.7. A perfect classification would result in an *Average Recall* of 74.7 as shown in Table 3.1. Even using ground truth instances which are directly presented to the *CNN* only reach an *Average Recall* of 55.4. This indicates, that the lightweight *CNN* is the performance bottleneck of the entire method, and not the cluster quality of the *FLIC*.

Runtime

The method presented in this section is composed of four main steps: clustering, data arrangement, prediction, and bounding box fitting. The data arrangement was not explicitly presented, as it consists of cropping the cluster instances and resizing them to 32×32 patches. The process is depicted in it's entirety in Figure 8.10. The runtime of the method was analyzed to evaluate its computational efficiency on a single Intel® Core™ i7-11850H @ 2.50GHz CPU core. The results are listed in Table 8.2.

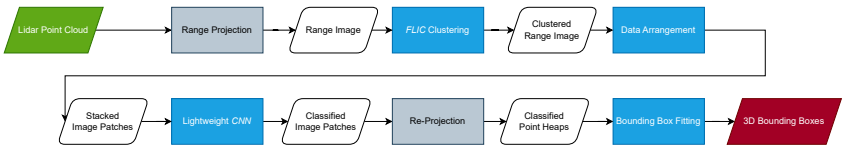

Figure 8.10.: Flowchart Illustrating the Steps Involved in the Object Detection Algorithm using Lidar Data. The algorithm begins with the raw lidar point cloud and progresses through various stages, ultimately leading to the generation of three-dimensional bounding boxes. Key operations are highlighted in blue, while data structures employed between steps are represented as rounded parallelograms. The range projection and re-projection steps are included for completeness but are depicted in gray to reflect their basic nature in the process.

The clustering step in this algorithm utilizes the *FLIC* algorithm, which exhibits a runtime complexity of $O(n_p)$, where n_p represents the number of points in the lidar data. The *FLIC* algorithm is essentially a traditional "*one component at a time*" *connected components labeling (CCL)* algorithm. It operates by visiting each pixel (lidar point in a range image) once and performing operations on neighboring pixels to establish connectivity and assign labels. Since every pixel is processed exactly once, the complexity remains linear in relation to the number of pixels, resulting in a complexity of $O(n_p)$ [2]. This step's purpose is to group the lidar points into clusters that correspond to different objects within the scene. On average, with four *Map Connections*, the mean runtime for a single point cloud is 30 ms, with a standard deviation of 2.82 ms.

The data arrangement step involves the transformation of the clustered lidar data into a cropped lidar image format suitable for the prediction step. This step has a runtime complexity of $O(n_c)$ as it involves iterating through the clusters n_c and arranging the data in a specific format. As this step was not optimized but ran in a *Python* loop, the mean runtime for a given lidar frame is 35 ms with a standard deviation of 3.39 ms.

The prediction step involves the use of a trained convolutional neural network to predict the class of each cluster sequentially on a CPU. The runtime of this step can be considered as $O(n_c)$, where m is the number of clusters. A mean prediction time of 15 ms with a standard deviation of 5.76 ms was measured for representative sets of clusters.

The final step is the fitting of bounding boxes around each predicted object. This step has a runtime complexity of $O(n_p)$ as it involves iterating over the predicted objects n_p and fitting a bounding box around each one. The measured mean runtime is 0.25 ms for a single point cloud with a standard deviation of 0.02 ms.

To determine the big O notation for the full sequence of all four steps, the worst-case scenario among the steps has to be considered. Since the steps are performed sequentially, the complexity of the most complex step defines the overall complexity [37]. Each step exhibits a linear complexity, therefore, the big O notation for the full sequence of all four steps is also linear $O(n)$, in which n is defined more general, as the complexity depends on the number sets of the sub-steps.

Table 8.2.: **Runtime Profiling of the Detection Method.** The method was profiled on an urban scene of 300 frames, in which at all times cars, pedestrians and cyclists are present. It is clear to see, that the two main components *FLIC* and the lightweight *CNN* only have a combined ratio of 0.56 of the total runtime, while the inefficient data arrangement has the largest individual share with 0.43. The time complexities are related to the number of lidar points n, the number of clusters m and the number of predicted instances of the four classes car, truck, bike and pedestrian, p.

Sub Method	Time (ms) ↓	Ratio	Complexity
FLIC Clustering	30	0.37	$O(n_p)$
Data Arrangement	35	0.43	$O(n_c)$
Lightweight *CNN*	15	0.19	$O(n_c)$
Bounding Box Fitting	0.25	0.00	$O(n_p)$
Total	81	1.0	$O(n)$

8.2.3 Conclusion

A novel method for online object detection using CPU-based panoptic lidar segmentation was presented in this section. A classification convolutional neural network was combined with a clustering technique and a cuboid fitting algorithm to segment a lidar point cloud in real-time and extract objects as three-dimensional cuboid detections from the environment.

The evaluation of this method shows promising results, but there is still room for improvement. Particularly the classification of the segmented clusters could be improved by a modern image classification network for better accuracy and inference speed [91]. Despite this, the visual inspection of the results shows good results, especially when combined with a tracking method which suppresses false positive predictions. The tracking is not included in this work. The method has been successfully implemented and tested as part of the publicly funded research project "*@CITY Automated Cars and Intelligent Traffic in the City: Subproject 7 - Interaction with weaker road users*".

In summary, this method offers a promising technique for online object detection utilizing lidar data. It has the potential to become an essential tool in autonomous systems, thanks to the benefits of real-time object detection on a single CPU core. With further research, this approach could lead to even better performance and practical applications.

8.3 Lidar Segmentation for Closed Loop Re-Simulation

The development and verification of modern Advanced Driver Assistance Systems (ADAS) can be a challenging task. It typically requires significant amounts of data to be recorded, stored, curated and annotated. All of which is then used to re-simulate the recorded data on the ADAS algorithms and software.

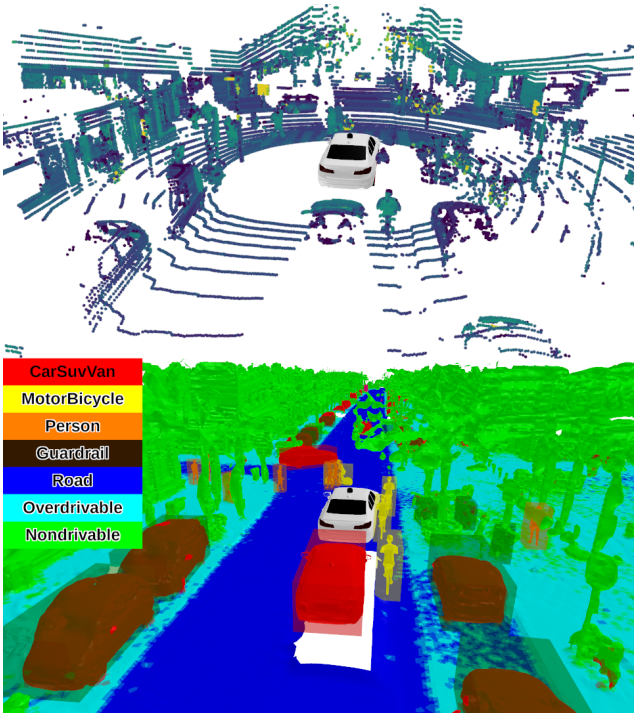

Figure 8.11.: **Re-Simulation Environment Created from Lidar Segmentation.** The entire environment is re-simulated (bottom) from the original lidar data (top).

The SAE (Society of Automotive Engineers) defines 5 levels for steps along the path to fully autonomous driving [@33] based on the level of control a driver has over the vehicle and the vehicle's ability to operate in different conditions. Level 0, no automation, where the driver controls the vehicle. Level 1, driver assistance, where automated systems assist in specific situations, e.g., adaptive cruise control and lane departure warning. Level 2, partial automation, where the vehicle can perform more complex functions such as following other vehicles, change lanes, and park itself under certain conditions, but requires the driver to be ready to take control at any time. Level 3, conditional automation, where drivers can disengage in specific situations and allow the car to take over driving duties in stop-and-go traffic, but requires the driver to remain alert and ready to take control. Level 4, high automation, where the vehicle's system can handle all driving functions for routine routes. And level 5, full automation, where the vehicle is fully autonomous and does not require a driver.

ADAS features of level 1 can be verified by replaying sensor inputs and measuring the performance of the systems for their specific use cases. However, for the development and verification of autonomous driving systems of level 2 and beyond, this approach is no longer valid. This is because such systems require active control of the vehicle, and previously recorded data cannot be used as a stimulus for the software stack. Closed-loop simulations are necessary, which provide a reactive input stimulus and accurately create artificial perception inputs, mimicking the actual sensor and actuator behavior. It is important to provide realistic, challenging, and diverse scenarios in the simulation to ensure the software performs well in all kinds of situations. The static scenery of the simulation must be rich, realistic, and diverse. It must accurately capture the variety of the real world. Dynamic content, such as other road users, must also be included and exhibit realistic and diverse behavior.

Traditional methods for creating simulations, such as hand-crafted scene modeling [71] and procedural generation based on mathematical rules [193], can be costly and time-consuming, and may struggle to cover the sheer variety of scenarios that need to be simulated.

A more efficient and effective approach is offered by the proposed solution of this section. It uses machine learning and data-driven techniques to generate and populate realistic and diverse dynamic environments for re-simulations.

8.3.1 Method

The method in this section is an alternative solution to a simulation-driven re-simulation. Instead of using a hand-crafted scene environment for the ADAS system, the variety of real-world scenarios is directly captured and turned into an accurate full-scale closed-loop re-simulation by recreating the real data in a simulation environment. This avoids the engineering effort of creating simulated environments from scratch, while increasing the scene's complexity and realism. The method is split into three parts. The recreation of the static environment, the population of the re-simulated scene with the original dynamic actors, and lastly the sensor re-simulation of novel viewpoints.

Static Environment

The recreation of the static environment is straight forward and a re-implementation of the scene reconstruction that was outlined in Chapter 7.2.2 and in Section 8.1. Once for the domain adaptation between different lidar sensors and once for the creation of automatic semantic grid map ground truth creation. Quickly summarized, the method works as follows. First, a real-world scene is captured by driving in the environment with a vehicle. This vehicle is equipped with a lidar scanner that records point cloud data of the surrounding.

The lidar frames are processed by three independent algorithm branches:

(a) An ensemble of multiple object detection networks [127, 178] for cars, pedestrians, two-wheelers and trucks. All combined and corrected by a Kalman filter [115].

(b) An ego-motion estimation algorithm that combines SLAM [177] and host vehicle velocity information, presented in Section 8.1.2.

(c) A high quality semantic segmentation network [228].

The lidar points of all dynamic objects are removed by using the tracked cuboid detections of (a) to select all points inside of the cuboids. The remaining, static points are accumulated to one large point cloud, by adjusting the relative position of each point cloud using the ego-motion estimation of (b). All points of the resulting global point cloud are already segmented by (c) to have a class label associated with each point. Last the global point cloud is processed with a Poisson surface reconstruction algorithm [116] to create a three-dimensional mesh from the point cloud. Each vertex and face of the mesh inherits the semantic segmentation of the global lidar points with a distance-wise and class-wise weighted influence of the ten closest points to each vertex.

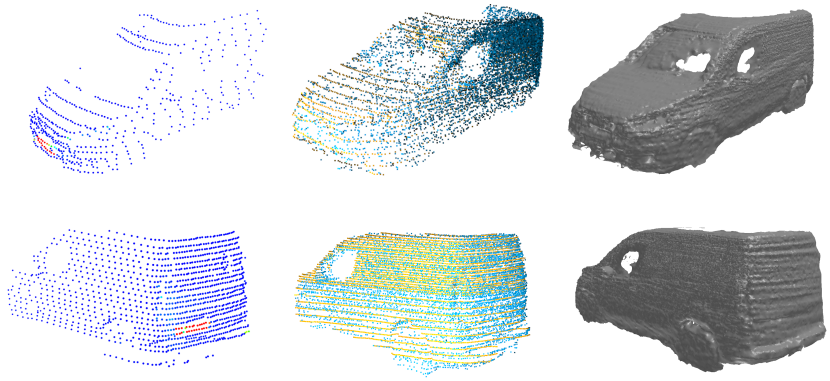

Figure 8.12.: **Mesh Recreation of Rigid Dynamic Objects.** The point clouds of the dynamic rigid road users are accumulated over the whole sequence (left) and registered via *point2plane ICP* [54] to each other (middle) in order to create a dense object point cloud which is used to recreate the object as a mesh (right) via the Poisson surface reconstruction algorithm [116].

With this, a fully reconstructed three-dimensional model of the static environment becomes available as a basis for the re-simulation. This recreation of the real-world data removes the need for manually designing this environment or generating it procedurally from a catalog of rules.

Dynamic Environment

The previously removed dynamic objects are used to re-populate the static scene. A random subset selection of the dynamic content is "replayed". By varying the randomly selected subset of objects which are instantiated, multiple variants of the scene can be created for re-simulation. All dynamic content is inherently consistent and meaningful as it is the true behavior observed in the real world. This removes the need for manually designing dynamic actors in the scene and controlling the behavior of these actors.

The actors are represented by three-dimensional meshes. Rigid dynamic objects, such as cars and trucks, can be restored as mesh objects in their original shape if they passed the recording vehicle during the original recording, or were overtaken so that the vehicles were recorded from multiple perspectives. The process of reconstructing a dynamic vehicle is as follows. All points included in each of the cuboids belonging to the same vehicle track are extracted and overlayed in a coordinate system which represents the relative position of each point in its respective cuboid.

These axis aligned cuboid point clouds are combined into one dense point cloud by *iterative closest point* alignment (*ICP*) [54]. For this, an efficient implementation of the *point2plane ICP* is used which is faster and more precise than the *point2point ICP* [35]. The normal vector of each point is calculated by using the range image representation. It is calculated twice for each point in the range image, each time by a cross product between the direction vectors from the point in question to two

of its neighboring points in the range image. The cross product is calculated between the northern and eastern neighbor and between the southern and western neighbor:

$$\mathbf{n}_{NE} = \mathbf{a}_N \times \mathbf{a}_E \tag{8.3}$$

$$\mathbf{n}_{SW} = \mathbf{a}_S \times \mathbf{a}_W, \tag{8.4}$$

where the two normal vectors $\mathbf{n}_{NE}$ and $\mathbf{n}_{SW}$ are averaged to obtain the final point-wise normal vector

$$\mathbf{n} = 0.5(\mathbf{n}_{NE} + \mathbf{n}_{SW}). \tag{8.5}$$

The *point2plane ICP* then follows the general steps as outlined in [54]. It iteratively finds the closest point on a reference plane to each point in the target point cloud, and adjusts the position of the target point cloud to minimize the distance between these closest points. This process is repeated until the alignment between the two point clouds is deemed sufficiently accurate. The process of accumulating point clouds of the same object via *point2plane ICP* is shown in Figure 8.12 (middle).

Usually, a vehicle can only be seen from three sides in a drive-by scenario. Nonetheless, vehicles are usually symmetrical. The point cloud can therefore be mirrored along the length axis of the bounding box to fill the missing side. The final object point cloud is then processed with the Poisson surface reconstruction algorithm [116] in order to recreate the surface of the rigid object which is then used as a mesh proxy in the re-simulation.

Opposed to rigid road users, non-rigid road users, such as pedestrians and cyclists, cannot be replicated from the point clouds. This is because the relative motion of limbs prevents the point clouds from accumulating. It would lead to an incorrect representation of these road users. Rigid actors, that are only partially visible can also not be reconstructed in their entirety. These road users are therefore represented by proxy meshes, i.e., pre-defined general shaped mesh objects of the given classes. These are scaled and oriented according to their bounding box parameters.

Re-Simulation

A single recorded log can thus provide a realistic static environment plus a wide variety of dynamic content within the scene. The original scene can be augmented to new scenarios by removing singular instances, or change the position of the ego vehicle in the recorded scene. To further increase the amount of diversity, the recreated scene can be used to define one of the actors as the ego-vehicle in the recorded log. The initial kinematic properties of the simulated agents can be altered within the simulation, e.g., the actual spawning position can be drawn from a Gaussian random distribution centered around the real-world target actor or slightly change the initial velocity. In short a single log of a couple of minutes can provide the source material for hundreds of realistic, meaningful and semantically correct augmented scenes for re-simulation.

The depth of the re-simulation can also be adjusted to the needs of the ADAS system. In the abstracted version, the true sensor inputs are not required, as the decision model is fed the cuboids as pseudo outputs of the perception stack. The quality of the detections can be adjusted to drop cuboids at random, add false positives or add noise to the position, orientation or velocity estimations. The

inputs to the decision model can be reproduced for the given simulation and evaluated for validity, by checking for intersections of the ego vehicle with objects and the static mesh world. This would indicate accidents caused by the system.

On top of the abstracted information of the static and dynamic content, the inclusion of the whole perception stack on which the decision model depends is also possible. Due to the holistic representation of the static scene as well as the dynamic actors for the entire recorded time frame, a complete picture of the entire scene at any point in time is recreated. The recreation of raw lidar data for the perception stack can be derived directly from the mesh environment. The mesh world, objects and proxy objects are regenerated in the structure of a defined lidar sensor by projecting the whole scenario for the given position in space and time. The process was already described in detail in Chapter 7 for the recreation of lidar data from a mesh world for domain adaptation between different sensors.

The resulting point cloud exhibits the same structure and orientation as the defined input sensor. The sensor parameters can also be adjusted, to recreate the raw sensor data for a different lidar sensor than the one that recorded the original data as shown in Chapter 7.

8.3.2 Evaluation

The performance of the presented re-simulation method could not be evaluated due to missing ground truth data. As such, the evaluation was carried out via a visual inspection of the results. As shown in Figure 8.13, a wide variety of scenarios can be be re-simulated ranging from urban inner city scenes, to large open road and even to parking garages.

The mesh environment and the mesh representations of the dynamic actors enable a rich and diverse set of re-simulations. The meshes are accurate reconstructions of the underlying real world environment. This can be seen in the direct comparison of the mesh environment and the original three-dimensional lidar measurements shown in Figure 8.11. These results suggest that the method is able to generate realistic and accurate synthetic data that enables a re-simulation of abstracted data as well as the perception stack due to the regeneration of sensor input data from the simulation.

8.3.3 Conclusion

This section presents a new method that uses machine learning and heuristic techniques to simulate ADAS features accurately. The method generates synthetic abstracted data for decision and path planning algorithms as well as sensor data for the perception stack of an ADAS system. At the time of writing a similar method to the one outlined in this section has been published by NVIDIA called *"NVIDIA DRIVE Sim"* [@154]. It can not, however, be directly compared to the performance of the presented method as it is not openly available. Future research will focus on testing and evaluating the potential of this method. Overall, this method shows promise in advancing the development of more efficient and precise re-simulation of ADAS features, which could be valuable for researchers and developers in this field.

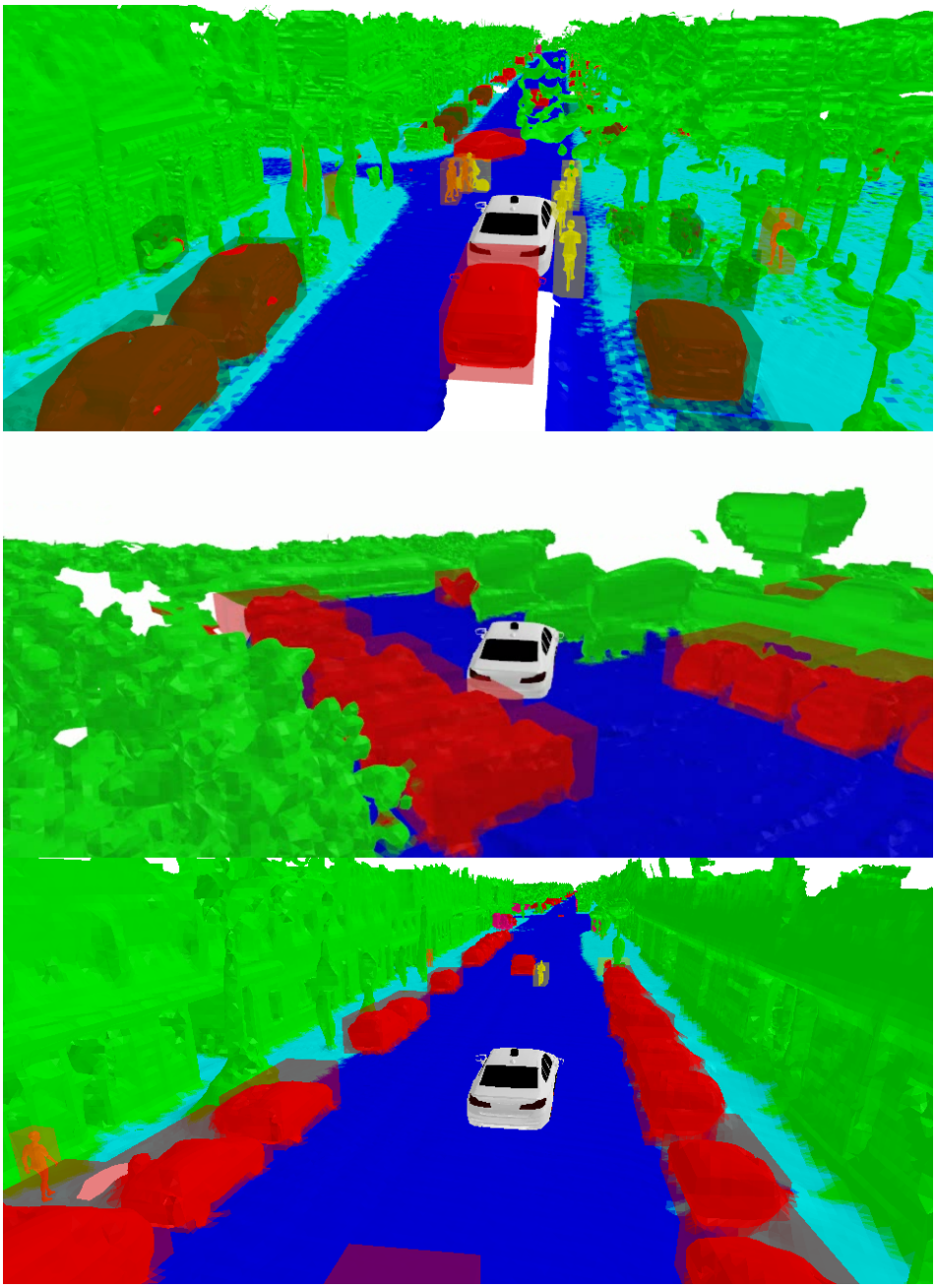

Figure 8.13.: **Various Reconstructed Scenes.** The presented method is not restricted to specific environments. The combination of a reliable ego-motion estimation with multiple object detection networks and a high quality semantic segmentation network enables the recreation of a wide variety of scenarios.

8.4 Lidar Segmentation Augmentation Techniques for Semi-Supervised Object Detection

The companies *Innoviz Technologies* and *NVIDIA* conducted a joint workshop at the *European Conference on Computer Vision 2022*. For this, they created a challenge [@112] in which they published a small dataset of the new *InnovizTwo* lidar sensor [@111]. The objective of the challenge was to develop an object detection model for a hidden dataset from the same lidar sensor. Participants were provided with 1,200 unannotated lidar scenes and 100 bounding box annotated scenes to train their networks. A promotional image of the InnovizTwo lidar data used in the challenge is shown in Figure 8.14. The ultimate aim was to create the most effective performing model for this specific task.

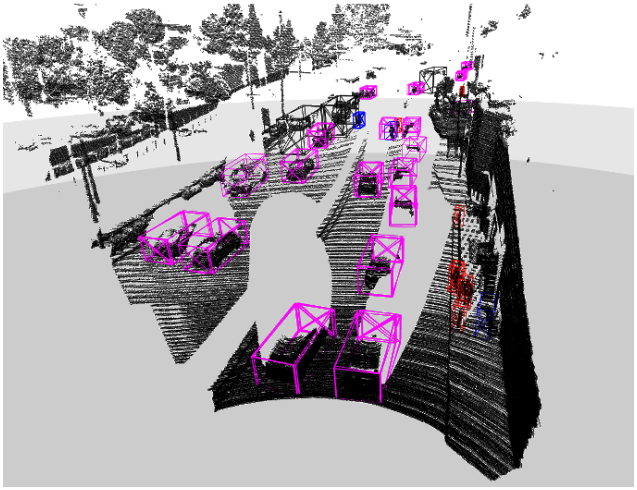

Figure 8.14.: **Prediction Result of the Method Outlined in this Section.** Visualisation of a single *InnovizTwo* lidar frame, that was part of the "The LiDAR Self-Supervised Learning Challenge: Learning From a Limited Amount of High-Resolution LiDAR data."[@112].

The augmentation methods presented in Chapter 6 were created for the improvement of networks for semantic segmentation of 360° rotating lidar sensor data. In [31] a simplified version of the method presented in Section 6.2.2 was used to improve three-dimensional bounding box detection networks with great success in quality and efficiency. An adaptation and combination of the methods of Chapters 6 and 7 seemed a promising solution to train a well-performing object detection model from the limited amount of data. In particular, they represented promising candidates, considering the evaluation of these methods for training networks for semantic segmentation with harshly reduced amount of data as shown in Section 6.3.3 and Section 7.3.

The method outlined in this section won the 1^{st} place[2] in the "The LiDAR Self-Supervised Learning Challenge: Learning From a Limited Amount of High-Resolution LiDAR data" of the *ECCV workshop on 3D Perception for Autonomous Driving* [@112].

8.4.1 Method

This section provides a brief summary of the reformulated augmentation and domain adaption methods of Chapters 6 and 7. The focus of the method lays on tuning and reformulating them for use with a solid-state lidar and for three-dimensional bounding box detection.The *OpenPCDet* [190]

[2] `https://eval.ai/web/challenges/challenge-page/1861/leaderboard/4375` (Please note that the ranking is listed in reverse order, with the first method listed last and the last method listed first.)

implementation of *PV-RCNN* [24] is used as a baseline and toolkit for development. The challenge organizers provided a forked version of this repository that was adjusted for the *InnovizTwo* data format [@112].

In an initial statistical analysis, the number of instances per class are summed and compared. In order to have a sufficiently large training and validation set, the 103 annotated lidar frames are split, so that both sets have a similar class distribution. The training set consists of 53 frames and the validation set of 50 frames.

The utilized method is split into four main aspects. Instance injections, mesh injections, scene fusion, and a novel use of pseudo labels. The main focus for the injection and fusion augmentations is to manipulate the data only in certain ways, such that the resulting scene could in fact be recorded by the sensor. In plain language, no manipulation is applied to the point cloud which the sensor would not be able to capture due to the underlying physical limitations such as resolution, range and occlusion. An example of a non-valid injection would be a pedestrian standing behind a high wall.

Instance Injection

To ensure a structurally intact instance injection, the objects are not simply considered as the contiguous contents of a box, but each lidar point belonging to the object is checked individually. The range image projection is an efficient method for this task [97]. The point clouds of the dataset are transformed into an image-like representation using a spherical coordinate transformation. Due to the discrete sampling of the individual lidar channels and the lack of intersections between them, each lidar point can be assigned to a fixed pixel within the range image. This allows for the injection of an object into the scene, either fully or partially, depending on whether the corresponding lidar point of the new object or an existing object within the original scene is closer to the lidar sensor. Projecting the data onto a range image representation reduces the dimensionality to two dimensions instead of three which is computationally more efficient. As a result, injections can be applied at training time with a dynamic re-sampling of instances, rather than using a fixed set of pre-augmented scenes. The chosen training data split is sampled for all bounding box labels and the points inside of those boxes. Thus a database is created, consisting of 439 cars, 19 pedestrians, 14 cyclists and 36 trucks. These objects can be injected only at two relative positions to the lidar sensor due to the underlying physical properties of the *InnovizTwo* lidar sensor. The original sampling position and the x-axis (left-right) mirror of the same. Any other shift or rotation of the instances would disrupt the structural integrity of the lidar point cloud and create a scene that the lidar sensor could not capture in reality. This is one of the main differences between this method and the rotating lidar scanners discussed in Chapter 6, as the rotation structure allowed for the rotation-independent injection of instances. The process of the instance injection is visualized in Figure 8.15. A pedestrian is sampled from the database (red) and injected into a lidar scene (blue). The occlusion of the new injection is highlighted in the point cloud (cyan shadow) and will be removed in order to preserve the lidar sensor structure.

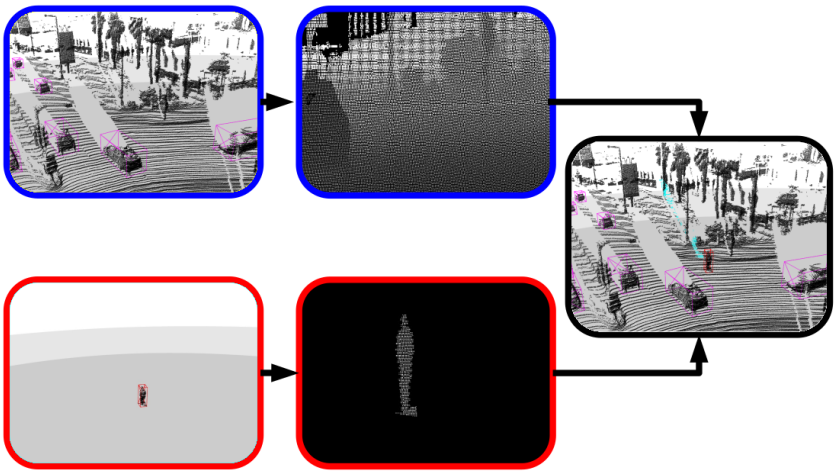

Figure 8.15.: ***InnovizTwo* Instance Injection.** The pedestrian is cropped from a different frame and injected via the structure aware point cloud injection method as outlined in Section 6.2.2. Both the lidar scene (blue branch) and the injection (red branch) are projected into the range image view and a point-wise range competition is applied to each point pair. The cyan lidar points represent the occlusion shadow of the pedestrian, that will be removed in order to produce a physically valid injection.

Mesh Injection

There are few examples of important classes such as pedestrians and cyclists in the training set of the *InnovizTwo* data. While re-injecting instances into more frames slightly alleviates this problem, it also carries the risk of overfitting on these few examples and decreasing performance on unseen examples. Pedestrians and cyclists are commonly included in object detection datasets, such as those in [86, 13, 50, 186, 117, 214]. Naive *Copy-Paste* injection methods [216, 21] can be used to incorporate these objects, but these approaches do not preserve the underlying lidar structure. The structure-preserving method presented in this section cannot be used to inject objects captured by a different type of lidar sensor, as this would lead to structural flaws in the resulting point cloud. It is also worth noting that using data from a different domain can negatively impact the performance of the network, as outlined in Chapter 7 with respect to semantic segmentation. Therefor synthetic data in the structure of the *InnovizTwo* is generated for these classes, to increase the injection data pool without introducing a domain shift.

The basic functionality and recording mode of the sensor [@111] is taken into account and used to recreate a virtual twin of the sensor as already done for semantic segmentation in Chapter 7. This includes re-engineering of the sensor origin, the relative position of each lidar measurement in the spherical coordinate system projection, and the reliable detection range from the available real data. This information is used to reformulate an orthogonal camera in the spherical coordinate system.

With this virtual twin sensor, open-source three-dimensional mesh models (see Figure 8.16) of pedestrians [@183, @163, @200] and bikes [@150, @151, @203, @73, @43] are recorded in the spherical coordinate system. The captured depth images are sub-sampled to the lidar sampling structure of the sensor and re-transformed to the corresponding Cartesian coordinates.

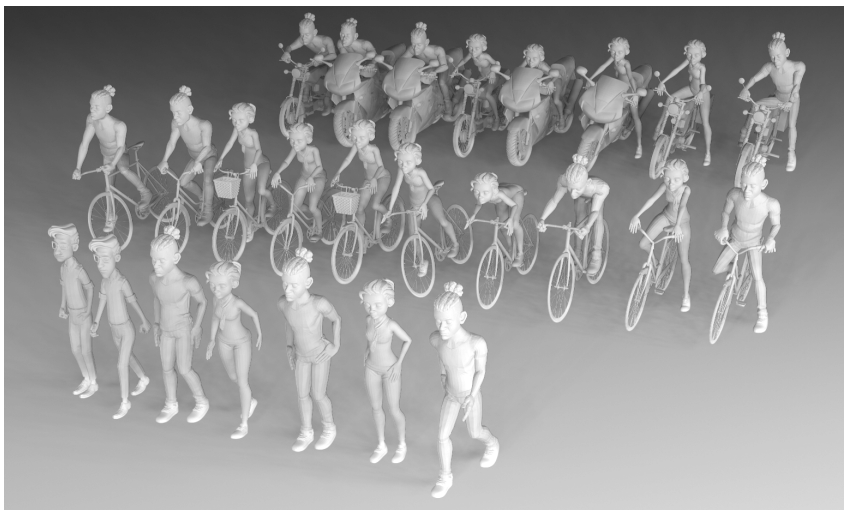

Figure 8.16.: **Three-Dimensional Mesh Models.** Royalty-free [@150, @151, @203, @73, @43] and creative common licensed three-dimensional mesh models [@183, @163, @200] are used in this work, that are re-posed and recombined in order to create diverse examples of underrepresented classes.

The resulting point cloud instances are saved in a second injection database. The process of sub-sampling the depth images to the lidar sensor structure is visualized in Figure 8.17. The resulting instance point clouds can be inserted with their bounding boxes into the scenes while keeping the structure of the lidar sensor intact as shown in Figure 8.18. The synthetic injections have the same structural properties as the original point clouds of the dataset. These mesh injections combine the *'real-to-real'* domain adaptation method of Chapter 7 with simulated mesh objects to a mixed *'simulation-to-real'* domain adaptation method.

Scene Fusion

Another challenge posed by the limited number of training scenes is the availability of negative examples for the neural networks, i.e., locations in the lidar frames that do not belong to the target classes. The more scenes that are available, whether full of objects or almost completely empty, the better object detection networks can distinguish between the desired classes and background objects that are similar in appearance. To address this, the structure-aware point cloud fusion method of Section 6.2.3 is adapted for use with the *InnovizTwo* lidar sensor with special care for the bounding box labels. In the original work outlined in Chapter 6, the fusion method could lead to merged objects, such as two cars overlapping each other. In order to prevent these unrealistic object mergings, the *Intersection over Union* of all cuboid labels of both parent point clouds are calculated in the birds eye view projection. If two boxes overlap each other with an $IoU > 0$, the cuboid label as well as the points inside the cuboid of the second point cloud are removed, to prevent them from merging.

The last step of the method compares two complete scenes point by point and generates new scenes by keeping only partial aspects of the two "parent" point clouds. With 53 originally available frames, a total of 53^2 = 2,809 fused frames can be created by combining each point cloud with every other, and even increased to 53^3 = 148,877 training frames by also adding an x-axis mirror of one of the

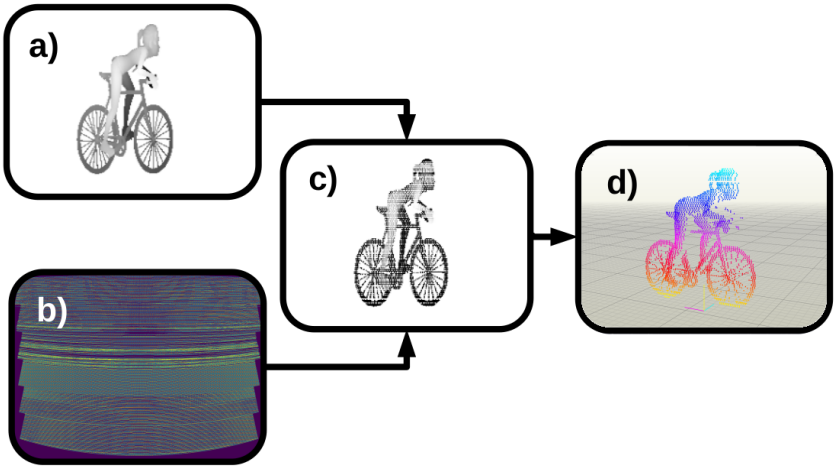

Figure 8.17.: **Mesh Objects are Retraced in the *InnovizTwo* Structure.** The models are captured by a virtual depth camera with the field of view of the sensor (a) the depth image is cropped to the instance for this visualization. The sensor's lidar tracing positions (b) are used to subsample the depth image to the individual lidar point positions (c) which are re-transformed to the three-dimensional Cartesian point coordinates (d) via their range, azimuth and elevation angle.

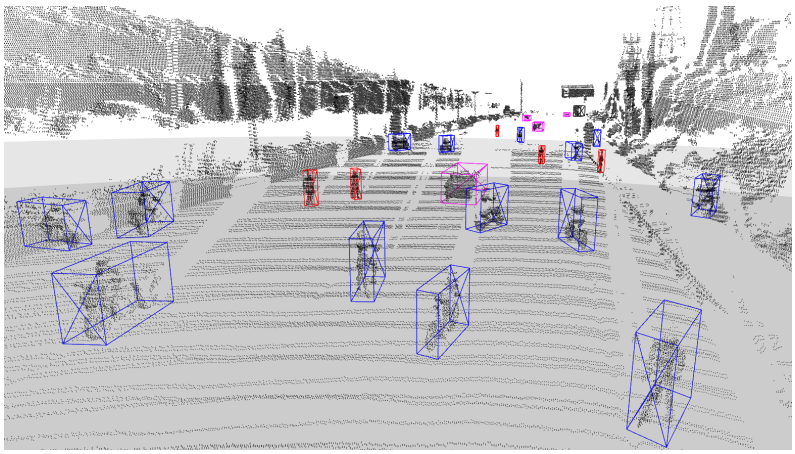

Figure 8.18.: **Synthetic Object Injected into Real *InnovizTwo* Data.** The meshes are retraced in multiple random positions and rotations, in order to inject the resulting point clouds of the underrepresented classes in the structure of the *InnovizTwo* sensor. The blue cuboids and their contents represent the inserted bicycles and motorcycles, while the red boxes are additionally created pedestrians. The purple boxes are cars of the original scene.

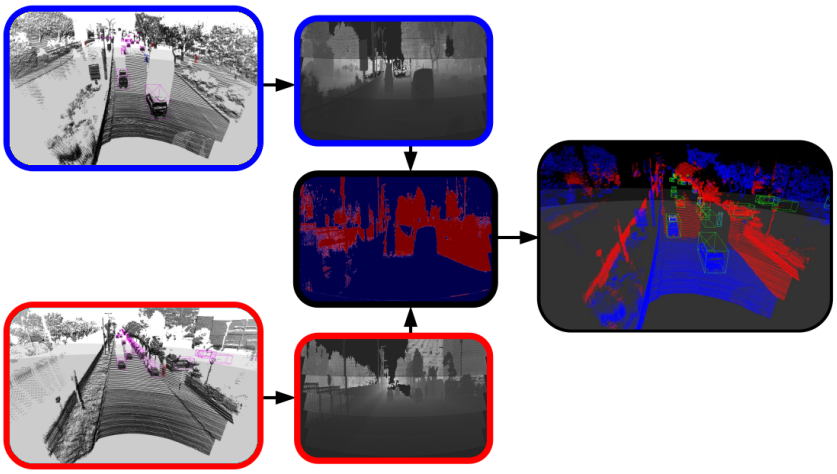

Figure 8.19.: **Scene Fusion Process on *InnovizTwo* Data.** Two randomly selected point clouds of the training dataset are combined by projecting them into the sensors field of view in a range image representation. The image domain allows for an efficient point-wise range competition between the two lidar frames in order to generate a new point cloud consisting of parts of each parent point cloud.

"parent" point clouds. This process can be further varied by adding a third point cloud to the fusion point cloud. The process for *InnovizTwo* lidar data is illustrated in Figure 8.19.

Semi-Supervised Pseudo Labels

Using the augmented data described earlier, a *PV-RCNN* network [24] is trained and subsequently applied to the 1,200 unlabeled frames in the dataset. The prediction labels generated by the network are then saved in a new dataset. Training directly on this new data would result in only marginal improvements, as the errors of the original network would be consolidated by this data. For this reason, the predictions are first processed before they are used as pseudo labels for training.

For each class at different distance bins, threshold scores are fine-tuned on the validation set, with the goal of keeping only detections which are certain to be correct. The defined thresholds are selected for each range and class bin to yield close to zero false positive detections on the validation set. A similar performance to the validation set is expected on the unlabeled data, therefore these thresholds are set to prevent the addition of too many false positives to the pseudo label training data.

Detections from the unlabeled data above the selected thresholds are extracted as additional new ground truth. All detections below these thresholds are removed from the new pseudo labeled data. Not only are the box labels removed, but also the corresponding lidar points inside the boxes. This is an important step, as the very high threshold values decide that many valid detections are not confident enough and therefore are not to be trusted as pseudo ground truth labels. Removing the cuboids while keeping the underlying lidar points would create false negatives.

This simple data cleaning enables the addition of pseudo ground truth labels into the injection database as well as using whole pseudo ground truth frames as additional scene fusion parent point

clouds. This combination of ground truth and pseudo ground truth boosts the *mean Average Precision* of the *PV-RCNN* network by 22.62 from 57.25 to 79.87, as can be seen in Table 8.3.

8.4.2 Evaluation

The performance of the *PV-RCNN* models on the validation data is evaluated via the *mean Average Precision* (*mAP*) metric. It is calculated by first computing the *Average Precision* (*AP*) for each class, and then taking the mean of all class-wise *APs*. The *AP* is calculated as the area under the precision-recall curve which is a graph showing the *precision* on the y-axis and the *recall* on the x-axis. The *precision* is the fraction of true positive detections out of all positive detections, and the *recall* is the fraction of true positive detections out of all ground truth instances.

The *Intersection over Union* (*IoU*) is used as a decision threshold whether a detected bounding box is a true positive or a false positive. The *IoU* is a measure of overlap between two bounding boxes. It is calculated as the ratio of the area of the intersection of the two bounding boxes divided by the area of their union. An $IoU > 0.5$ between a ground truth cuboid of a given class and a prediction of the same is defined a true positive for the *AP*.

The equation for calculating the *AP* is

$$AP = \sum_{k=1}^{n} (R_k - R_{k-1}) P_k, \tag{8.6}$$

where R_k is the recall at the k^{th} point along the precision-recall curve, P_k is the precision at the k^{th} point, and n is the total number of sample points on the precision-recall curve. The mAP is calculated as

$$mAP = \frac{1}{C} \sum_{c=1}^{C} AP_c, \tag{8.7}$$

where C is the number of classes and AP_c is the *AP* of the c^{th} class. The *precision* and the *recall* are collected at 41 sampling points along the precision-recall curve to average the class-wise *AP*. In summary, the *mAP* is a metric that combines the precision and recall of an object detection algorithm across multiple classes. It is used to evaluate the performance of the object detection models in this section.

To compare the presented method to a baseline and to measure the influence of all parts of the method, three versions of the same network were trained. One on the original data, one on the augmented data and one version on the data with the augmentations and the proposed pseudo label additions. All networks were trained on a single *NVIDIA RTX 2080 Ti* for 1000 epochs, a total of 53,000 iterations, with a batch size of one, and a one-cycle learning rate scheduler [182] with a maximum rate of 0.01. All three trained models were evaluated using the *mean Average Precision* metric.

Incorporating additional augmentations and pseudo labels has a significant impact on the performance of the proposed methods, as demonstrated by the increase in the *mAP* from 55.97 to 57.25, and ultimately reaching 79.87, as shown in Table 8.3.

Figure 8.20.: **Validation Set Predictions of the Three Evaluation Models.** The network predictions of cars, pedestrians, bikes and trucks are shown as purple, red, blue and black boxes, respectively. The ground truth cuboids are colored yellow. All three networks successfully detect the car instances. Only the models trained using the augmentation methods successfully detect the pedestrian on the left and the bicycle on the far right. The model trained with the augmentation methods and additional pseudo labels correctly detects and predicts the truck in the background. The model trained with just the augmentations falsely classifies it as a car. The model trained on the original data does not detect it at all.

Table 8.3.: **Mean Average Precision and Average Precision per Class.** The performance of the baseline trained on the original data is compared to the same network trained with the presented augmentation methods, and with additional pseudo labels.

Method	$mAP \uparrow$	$AP_{Car} \uparrow$	$AP_{Ped.} \uparrow$	$AP_{Cyc.} \uparrow$	$AP_{Truck} \uparrow$
Baseline	55.97	**93.32**	0.00	33.33	**97.22**
Augmentations	57.25	91.47	8.33	43.06	86.14
Augs. + Pseudo Labels	**79.87**	90.41	**75.00**	**61.11**	92.96

The baseline model, which was trained without the augmentations and pseudo labels, has the highest *AP* for the car and truck classes, with values of 93.32 and 97.22, respectively. However, it exhibits the worst performance on the pedestrians and cyclists classes, with values of 0.00 and 33.33, respectively. The augmentation methods and mesh injections of the simulation domain adaptation improve the performance on the pedestrians and cyclists classes to 8.33 and 43.06, respectively, while reducing the *AP* on cars and trucks to 91.47 and 86.14, respectively.

The third network, which was trained on the augmented data as well as the pseudo labeled data, reaches the best performance for pedestrians with an *AP* of 75.00 and for cyclists with an *AP* of 61.11. The performance for cars is slightly lower than the previous two networks with an *AP* of 90.41, and the network reaches a *AP* of 92.96 for trucks, slightly less than the baseline model but better than the augmentation model.

In conclusion, the effectiveness of the proposed methods for improving object detection in challenging scenarios is highlighted by the results obtained. However, it is important to view the results with some caution due to the limited number of samples in the validation set, which consisted of only 50 lidar frames, with only 351 cars, 41 trucks, 11 pedestrians, and a total of 11 cyclists. To address this limitation, a qualitative evaluation via visual inspection was performed by inferring on the entire validation set. The model trained with the augmentation methods was found to demonstrate a visibly better performance compared to the model trained solely on the training set. The best visible performance was exhibited by the model trained on both the augmented data and pseudo labels.

For the ECCV challenge a different metric was used to measure the performance of the final submission. The evaluation was extended to additionally measure the performance of the three trained models on the validation set using the *average Intersection over Union* in the xy-coordinate plane

$$IoU_{XY,\,\mu} = 0.5 * IoU_{XY,\,\mu,\,GT\,vs\,Det} + 0.5 * IoU_{XY,\,\mu,\,Det\,vs\,GT}, \tag{8.8}$$

where $IoU_{XY,\,\mu,\,GT\,vs\,Det}$ is the average *IoU* of each ground truth annotations best fitting detection, and $IoU_{XY,\,\mu,\,Det\,vs\,GT}$ is the average *IoU* of each detections best fitting ground truth annotation. It is noteworthy that the $IoU_{XY,\,\mu}$ of the *PV-RCNN* model, trained with the augmentation methods and fused pseudo labels, performed worse than the baseline network trained on the raw data. This is due to the fact that the car class is heavily over-represented in the validation set, and the challenge metric does not evaluate class-wise, but rather averages over all instances regardless of the class. The *mAP* metric and visual inspection of the results were used for the development of the presented method. The $IoU_{XY,\,\mu}$ was ignored in the development and only used as a target metric for the final submission in the challenge.

Table 8.4.: **Class Agnostic Intersection over Union.** The three evaluation models are compared with the metric used in the challenge.

Method	$IoU_{XY,\,\mu}\uparrow$	$IoU_{XY,\,\mu,\,GT\,vs\,Det}\uparrow$	$IoU_{XY,\,\mu,\,Det\,vs\,GT}\uparrow$
Baseline	**0.60**	0.49	**0.71**
Augmentations	0.50	0.69	0.31
Augs. + Pseudo	0.51	**0.70**	0.31

Table 8.5.: **Submission Results of the Challenge.** Five teams submitted their results for the test set of the challenge.

Team Name	$IoU_{XY,\,\mu}\uparrow$
Gott	0.36
DUTH-VCG	0.48
dgist-cvlab	0.61
BITvisionLab	0.66
FrederikH	**0.68**

8.4.3 Submission

The evaluation metric of this challenge was $IoU_{XY,\,\mu}$, where for each annotation the best fitting prediction is taken into account ($IoU_{XY,\,\mu,\,GT\,vs\,Det}$) and vice versa ($IoU_{XY,\,\mu,\,Det\,vs\,GT}$) as seen in Equation 8.8. Based on this metric the *PV-RCNN* models were tuned in order to boost the performance on the challenge metric. This decision led to the use of an ensemble of multiple *PV-RCNN* models instead of a single one, and they were each trained for a total of 265,000 iterations. Three different voxel grids were used, as the performance of the classes and range bins was different depending on the grid size and resolution. The performance for every class and every range bin ($0-50\,\text{m}$, $50-100\,\text{m}$, and $>100\,\text{m}$) was evaluated in order to select three models, that had the best combination performance for all 12 range/class bins.

For all three models, the detections were kept without any confidence threshold, to keep the hard and uncertain detections of pedestrians and cyclists, as well as far range detections of all classes. These three networks detected more of the ground truth instances than a single model was able to detect, but they also introduced more false positives. Therefore an extension of the model ensemble with 5 additional checkpoints for each of the three voxel grids was added, in which the minimum confidence was raised to very high values, in order to only keep the predictions that were very certain to be correct.

This ensemble of a total of 18 models enabled the detection of very hard ground truth instances such as far away cyclists and pedestrians, while at the same time very confidently detecting easy targets, i.e., close cars. This combination increased the metric of $IoU_{XY,\,\mu}$ two-fold.

The first part of Equation 8.8, $IoU_{XY,\,\mu,\,GT\,vs\,Det}$, compares each GT cuboid only to the best matching prediction and ignores all others. The three networks that were not limited with confidence thresholds, detected very difficult objects, which raised this part of the equation. The false positives are no concern for this part of the equation. The additional confident networks improved the IoU of the

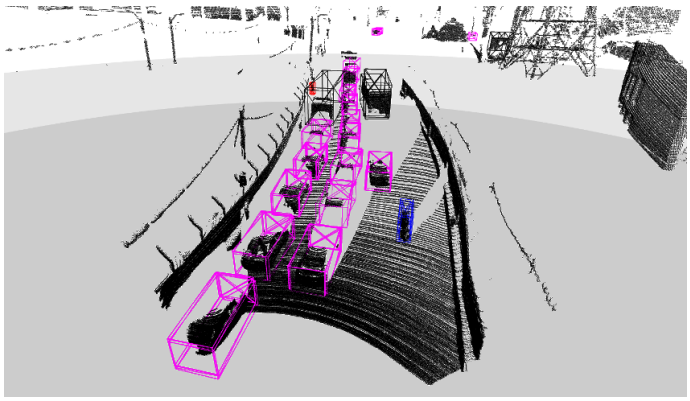

Figure 8.21.: **Combined Submission Predictions.** The results of multiple *PV-RCNN* models trained on three different voxel grid resolutions are combined to boost the challenge metric on the test set.

easier targets by offering multiple predictions for the same ground truth which raised the probability of generating a very well fitting detection.

The second part of Equation 8.8, $IoU_{XY,\,\mu,\,DetvsGT}$, compares each prediction cuboid with their best matching GT cuboid, even if another prediction already matches the same GT cuboid. By using multiple confident model outputs, the $IoU_{XY,\,\mu,\,DetvsGT}$ term of the equation was further raised, as multiple predictions boxes for easy targets fit the GT cuboids relatively well. This raises the average which minimizes the influence of any false positives from the three non-threshold networks.

Thus both parts of the equation for the final metric were raised by the proposed ensemble. The final results for this challenge had an $IoU_{XY,\,\mu}$ score of 0.68, higher by 0.02 than the next best submission as shown in Table 8.5. An example of a submission frame which combines the output of all sub-models can be seen in Figure 8.21

8.4.4 Conclusion

The use of multiple novel methods, including data augmentation, domain adaptation, and semi-supervised learning via pseudo labels, was described in this chapter for the purpose of training a well performing neural network for lidar object detection. This method was outlined in detail, and the evaluation shows the value of each part of the method, with impressive results being achieved. The method achieved the first place of the "The LiDAR Self-Supervised Learning Challenge: Learning From a Limited Amount of High-Resolution LiDAR data." of the *ECCV 2022 workshop on 3D Perception for Autonomous Driving* [@112].

8.5 Conclusions on the Versatility and Effectiveness of Lidar Segmentation in Autonomous Vehicles

The potential of lidar segmentation for enhancing the capabilities of autonomous vehicles has been highlighted in this chapter through the presentation of four different novel applications for automotive use cases.

The first method utilizes lidar segmentation, object detection, and heuristic algorithms to create semantic grid map data for the training of automotive online segmentation algorithms. This approach offers a more efficient and effective way of training algorithms, enabling them to better detect objects and navigate the environment.

The second method employs panoptic segmentation to create an online lidar detection algorithm that runs on a single CPU core. This method has demonstrated promising results for real-time object detection and classification, which is critical for the safety of autonomous vehicles.

The third method introduces a novel re-simulation environment that enables the training and testing of ADAS and AD software stacks. This environment facilitates the development and testing of more advanced and complex software, thereby accelerating the progress towards fully autonomous vehicles.

The fourth method uses lidar segmentation augmentation and domain adaptation techniques to train a challenge-winning semi-supervised object detection network. This approach has shown great promise in improving the accuracy and efficiency of object detection, especially when dealing with limited labeled data.

These methods illustrate the versatility and effectiveness of lidar segmentation in the automotive industry, showcasing how this technique can be applied in various ways to enhance the capabilities of autonomous vehicles. The applications presented in this chapter are just a few examples of the wide range of possibilities that exist for lidar segmentation, and it is exciting to envision the future advancements that will emerge as this technology continues to evolve.

9 Conclusion and Outlook

In this final chapter, the main research findings presented in this dissertation will be synthesized by providing a comprehensive summary of the key points and their implications. The broader significance of the research within the field will also be discussed, highlighting the research limitations and offering recommendations for future work, specifically in the area of lidar segmentation for autonomous vehicles.

To reiterate, the main research question addressed by this dissertation is how lidar segmentation can be utilized to enhance the capabilities and improve the safety of autonomous vehicles. The dissertation presents novel approaches for generating segmentation labels for lidar data, as well as methods for improving the training and robustness of these algorithms. Additionally, it introduces various cutting-edge applications of lidar segmentation in the automotive sector.

The research is divided into three main parts, with each part aiming to answer a different variation of the main research question.

In Part I, multiple novel approaches for instance, semantic, and panoptic segmentation are presented. These aim to answer the first variation of the main research question of this thesis: How can lidar segmentation be enhanced?

Chapter 3 of Part I presents the *FLIC* algorithm, which achieves real-time instance segmentation of lidar sensor data by reformulating the clustering of a three-dimensional point cloud to a two-dimensional *connected-components-labeling* problem. It exploits the underlying sensor structure as a range image to connect points by their three-dimensional distance to each other. To make the *FLIC* more robust against over-segmentation, *Map Connections* were added to the algorithm to bridge partial occlusions. The approach was evaluated on multiple datasets and not only was faster than comparable state-of-the-art methods but also provided significantly better instance segmentation quality.

Chapter 4 of Part I presents the *RangePillars* network, which functions as an additional module to be used with projection networks for semantic segmentation of lidar data. The *RangePillars* network encodes the three-dimensional lidar point cloud to a spherical voxel grid aligned with the range image structure of the chosen two-dimensional projection network for semantic segmentation. The main contribution of the *RangePillars* network is that it improves the predicted segmentation labels by fusing the three-dimensional geometric information with the range image networks pixel features for a more precise separation between objects. The *RangePillars* network was evaluated on the *SemanticKITTI* dataset and achieved a higher *mIoU* than previous methods that improve the three-dimensional labels of two-dimensional projection networks.

Chapter 5 of Part I presents two novel methods for lidar panoptic segmentation: *Lidar Cluster Classification* and *Lidar Image Panoptic Segmentation*. The former method combines the *FLIC*

with an additional lightweight *CNN* that classifies the clusters to a specific set of classes. The latter method combines existing lidar semantic segmentation networks with additional instance segmentation achieved with the *FLIC*. A novel *FLIC++* algorithm is also introduced that improves the *FLIC* via a parallelized approach. Both *Lidar Cluster Classification* and *Lidar Image Panoptic Segmentatio*n were evaluated on the *SemanticKITTI* dataset for panoptic segmentation. The former achieves real-time panoptic segmentation on a single CPU core, and the latter achieves a place in the top three methods for lidar panoptic segmentation on the dataset.

In Part II the main research question of this thesis is presented from a practical standpoint: How can the training cost of lidar segmentation be reduced?

Instead of developing new neural networks or increasing investment in gathering and labeling additional training data, existing semantic segmentation models are improved using the *Structure Aware Point Cloud Augmentation (SAPCA)* approach presented in Chapter 6 of Part II. This approach employs novel lidar-centric augmentation methods that improve all key metrics of semantic segmentation, and is more effective than increasing the amount of data by a factor of ten. The approach was evaluated on the *SemanticKITTI* dataset as well as the closed source Aptiv panoptic segmentation dataset. The augmentations improved all five neural networks they were applied to. Furthermore, the presented method outperformed the current State of the Art for semi-supervised lidar semantic segmentation.

In Chapter 7 of Part II, the augmentation methods of the previous chapter are revisited for training semantic segmentation models beyond a single dataset and lidar sensor. Neural networks for lidar semantic segmentation often suffer from poor generalization performance when trained on a single domain, as lidar data can vary greatly between different domains, such as different sensor types, environments, or seasons. This issue is addressed by introducing a new method for domain adaptation for lidar segmentation, aligning different lidar domains using sensor-aware domain adaptation modules. Self- and semi-supervised fusion methods at the data level enable effective training of lidar segmentation networks. The chapter provides a thorough evaluation and comparison to state-of-the-art methods on multiple real world datasets, demonstrating significant improvement caused by the novel domain adaptation methods.

By proposing novel approaches and techniques such as lidar augmentation, domain adaptation, and self-supervised learning, the two chapters answer the question of how to reduce the training cost for lidar segmentation. The techniques enable the training of well-performing networks using minimal amounts of labeled data frames, while improving the performance and generalization of the segmentation networks.

The last part of this thesis, Part III aimed to answer the last formulation of the main research question: What are feasible use cases of lidar segmentation for autonomous vehicles?

In Chapter 8 of Part III, multiple feasible use cases are explored. The chapter provides examples of diverse novel applications of lidar segmentation in the automotive industry, including creating semantic grid map data, developing an online lidar detection algorithm, establishing a novel re-simulation environment for ADAS and AD software stack training and testing, and training a semi-supervised object detection network using lidar segmentation augmentation and domain adaptation

techniques. These methods demonstrate the usefulness and effectiveness of lidar segmentation in enhancing the capabilities of autonomous vehicles.

While this dissertation has introduced innovative lidar segmentation algorithms and models, as well as new data augmentation and domain adaptation techniques, there are still several limitations in current approaches that need to be addressed. These include integrating lidar segmentation with other sensors, enhancing robustness to adverse weather conditions, and achieving real-time implementation on low-power and low-cost hardware platforms. Future research should focus on these areas to further advance the field and expedite the deployment of autonomous vehicles in various domains.

The presented work has advanced the State of the Art for lidar segmentation, as evidenced by the first-place award in a conference challenge[1], top scores in public benchmarks[2][3], and a best student paper award[4]. These advancements have been applied in various applications that improve the capabilities and safety of autonomous vehicles.

As a result, I hope that this dissertation inspires further research in the field of lidar segmentation and contributes to the ongoing efforts to make autonomous vehicles safer and more efficient. It has been an honor to share this work, and I hope it piques the reader's interest and encourages them to contribute to this exciting and rapidly evolving field.

[1] https://eval.ai/web/challenges/challenge-page/1861/leaderboard/4375
[2] https://paperswithcode.com/sota/semi-supervised-semantic-segmentation-on-24
[3] https://codalab.lisn.upsaclay.fr/competitions/7092#results
[4] https://icpram.scitevents.org/PreviousAwards.aspx#2021

Bibliography

[1]Martín Abadi, Ashish Agarwal, Paul Barham, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015 (cit. on p. 66).

[2]Ayman AbuBaker, Rami Qahwaji, Stan Ipson, and Mohmmad Saleh. "One scan connected component labeling technique". In: *2007 IEEE International Conference on Signal Processing and Communications*. IEEE. 2007, pp. 1283–1286 (cit. on pp. 35, 128).

[3]Eren Erdal Aksoy, Saimir Baci, and Selcuk Cavdar. "Salsanet: Fast road and vehicle segmentation in lidar point clouds for autonomous driving". In: *arXiv preprint arXiv:1909.08291* (2019) (cit. on pp. 47, 48).

[4]Alexandre Rame et al. "OMNIA Faster R-CNN: Detection in the wild through dataset merging and soft distillation". In: *arXiv preprint arXiv:1812.02611* (2018) (cit. on p. 81).

[5]Alexey Nekrasov et al. "Mix3D: Out-of-Context Data Augmentation for 3D Scenes". In: *International Conference on 3D Vision (3DV)*. 2021 (cit. on pp. 80, 82, 100).

[6]Alonso Inigo et al. "3d-mininet: Learning a 2d representation from point clouds for fast and efficient 3d lidar semantic segmentation". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5432–5439 (cit. on pp. 47–49, 60).

[7]Andres Milioto et al. "Rangenet++: Fast and accurate lidar semantic segmentation". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 4213–4220 (cit. on pp. 1, 24, 47, 52, 57, 59, 62, 80, 88, 89).

[8]Berman Maxim et al. "The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4413–4421 (cit. on pp. 53, 54).

[9]Brekke Åsmund et al. "Multimodal 3d object detection from simulated pretraining". In: *Symposium of the Norwegian AI Society*. Springer. 2019, pp. 102–113 (cit. on p. 81).

[10]Christopher Choy et al. "4d spatio-temporal convnets: Minkowski convolutional neural networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3075–3084 (cit. on pp. 24, 46).

[11]Dhruv Mahajan et al. "Exploring the limits of weakly supervised pretraining". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 181–196 (cit. on p. 47).

[12]Hanqi Zhu et al. "VPFNet: Improving 3D Object Detection with Virtual Point based LiDAR and Stereo Data Fusion". In: *arXiv preprint arXiv:2111.14382* (2021) (cit. on p. 82).

[13]Holger Caesar et al. "nuscenes: A multimodal dataset for autonomous driving". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11621–11631 (cit. on pp. 12, 79, 97, 100, 138).

[14]Hongyi Zhang et al. "mixup: Beyond Empirical Risk Minimization". In: *International Conference on Learning Representations*. 2018 (cit. on pp. 80, 82).

[15]Hugues Thomas et al. “Kpconv: Flexible and deformable convolution for point clouds”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 6411–6420 (cit. on pp. 46, 47, 57, 59, 62, 73, 80, 87–89).

[16]Jianyun Xu et al. “Rpvnet: A deep and efficient range-point-voxel fusion network for lidar point cloud segmentation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 16024–16033 (cit. on pp. 60, 82).

[17]Jinlai Zhang et al. “PointCutMix: Regularization Strategy for Point Cloud Classification”. In: *arXiv preprint arXiv:2101.01461* (2021) (cit. on pp. 80, 82).

[18]Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778 (cit. on p. 47).

[19]Larissa T Triess et al. “Scan-based Semantic Segmentation of LiDAR Point Clouds: An Experimental Study”. In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2020, pp. 1116–1121 (cit. on pp. 48, 87).

[20]Lue Fan et al. “Rangedet: In defense of range view for lidar-based 3d object detection”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 2918–2927 (cit. on p. 82).

[21]Martin Alsfasser et al. “Exploiting polar grid structure and object shadows for fast object detection in point clouds”. In: *Twelfth International Conference on Machine Vision (ICMV 2019)*. Vol. 11433. International Society for Optics and Photonics. 2020, 114330G (cit. on pp. 81, 138).

[22]Ran Cheng et al. “2-s3net: Attentive feature fusion with adaptive feature selection for sparse semantic segmentation network”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 12547–12556 (cit. on p. 46).

[23]Sangdoo Yun et al. “Cutmix: Regularization strategy to train strong classifiers with localizable features”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 6023–6032 (cit. on pp. 80–82).

[24]Shaoshuai Shi et al. “Pv-rcnn: Point-voxel feature set abstraction for 3d object detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10529–10538 (cit. on pp. 124, 126, 127, 137, 141).

[25]Venice Erin Liong et al. “AMVNet: Assertion-based Multi-View Fusion Network for LiDAR Semantic Segmentation”. In: *arXiv preprint arXiv:2012.04934* (2020) (cit. on p. 81).

[26]Whye Fong et al. “Panoptic nuscenes: A large-scale benchmark for lidar panoptic segmentation and tracking”. In: *arXiv preprint arXiv:2109.03805* (2021) (cit. on pp. 97, 103).

[27]Xu Yan et al. “Sparse Single Sweep LiDAR Point Cloud Segmentation via Learning Contextual Shape Priors from Scene Completion”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 4. 2021, pp. 3101–3109 (cit. on p. 81).

[28]Yunlu Chen et al. “Pointmixup: Augmentation for point clouds”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer. 2020, pp. 330–345 (cit. on pp. 80, 82).

[29]Stefano Allegretti, Federico Bolelli, Michele Cancilla, and Costantino Grana. “A block-based union-find algorithm to label connected components on GPUs”. In: *International Conference on Image Analysis and Processing*. Springer. 2019, pp. 271–281 (cit. on p. 35).

[30]Inigo Alonso et al. “Domain adaptation in LiDAR semantic segmentation by aligning class distributions”. In: *arXiv preprint arXiv:2010.12239* (2020) (cit. on pp. 2, 95, 96).

[31]Martin Alsfasser. “Contributions to Tracking and Artificial Intelligence Based Lidar Signal Processing for Automotive Applications”. Dissertation. Bergische Universität Wuppertal, 2022 (cit. on pp. 43, 136).

[32]Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. “OPTICS: ordering points to identify the clustering structure”. In: *ACM Sigmod record* 28.2 (1999), pp. 49–60 (cit. on p. 30).

[34]Octavio Arriaga, Matias Valdenegro-Toro, and Paul Plöger. “Real-time convolutional neural networks for emotion and gender classification”. In: *arXiv preprint arXiv:1710.07557* (2017) (cit. on pp. 64, 65).

[35]K. S. Arun, T. S. Huang, and S. D. Blostein. “Least-Squares Fitting of Two 3-D Point Sets”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-9.5 (1987), pp. 698–700 (cit. on p. 132).

[36]Mehmet Aygun, Aljosa Osep, Mark Weber, Maxim Maximov, Cyrill Stachniss, Jens Behley, and Laura Leal-Taixé. “4d panoptic lidar segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 5527–5537 (cit. on pp. 62, 73).

[37]Paul Bachmann. *Zahlentheorie: th. Die analytische Zahlentheorie (1894)*. Vol. 2. BG Teubner, 1894 (cit. on p. 128).

[38]Hernán Badino, Uwe Franke, and David Pfeiffer. “The stixel world-a compact medium level representation of the 3d-world”. In: *Joint Pattern Recognition Symposium*. Springer. 2009, pp. 51–60 (cit. on p. 116).

[39]Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. “SemanticKITTI: A dataset for semantic scene understanding of lidar sequences”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 9297–9307 (cit. on pp. 12, 40, 55, 62, 65, 79, 87, 90, 91, 93, 97, 103, 121, 126).

[40]Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. “Cython: The best of both worlds”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 31–39 (cit. on p. 59).

[41]Jon Louis Bentley. “Multidimensional binary search trees used for associative searching”. In: *Communications of the ACM* 18.9 (1975), pp. 509–517 (cit. on p. 30).

[42]Borna Bešić, Nikhil Gosala, Daniele Cattaneo, and Abhinav Valada. “Unsupervised domain adaptation for lidar panoptic segmentation”. In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 3404–3411 (cit. on pp. 2, 95, 96, 103).

[44]Frank Bieder, Sascha Wirges, Johannes Janosovits, Sven Richter, Zheyuan Wang, and Christoph Stiller. “Exploiting multi-layer grid maps for surround-view semantic segmentation of sparse LiDAR data”. In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2020, pp. 1892–1898 (cit. on pp. 115, 116).

[45]Igor Bogoslavskyi and Cyrill Stachniss. “Fast range image-based segmentation of sparse 3D laser scans for online operation”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 163–169 (cit. on pp. 1, 2, 30, 34, 35, 37, 39–43).

[46]Johann Borenstein, Yoram Koren, et al. “Histogramic in-motion mapping for mobile robot obstacle avoidance”. In: *IEEE Transactions on robotics and automation* 7.4 (1991), pp. 535–539 (cit. on p. 116).

[47]Shubhankar Borse, Ying Wang, Yizhe Zhang, and Fatih Porikli. “Inverseform: A loss function for structured boundary-aware segmentation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 5901–5911 (cit. on p. 96).

[48]Markus Braun, Fabian B Flohr, Sebastian Krebs, Ulrich Kreße, and Dariu M Gavrila. “Simple Pair Pose-Pairwise Human Pose Estimation in Dense Urban Traffic Scenes”. In: *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2021, pp. 1545–1552 (cit. on p. 124).

[49]Martin Brossard, Axel Barrau, and Silvère Bonnabel. “AI-IMU dead-reckoning”. In: *IEEE Transactions on Intelligent Vehicles* 5.4 (2020), pp. 585–595 (cit. on p. 118).

[50]Keenan Burnett, David J Yoon, Yuchen Wu, et al. “Boreas: A Multi-Season Autonomous Driving Dataset”. In: *arXiv preprint arXiv:2203.10168* (2022) (cit. on pp. 2, 95, 138).

[51]Rich Caruana. “Learning many related tasks at the same time with backpropagation”. In: *Advances in neural information processing systems* 7 (1994) (cit. on p. 96).

[52]Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. “Encoder-decoder with atrous separable convolution for semantic image segmentation”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 801–818 (cit. on p. 21).

[53]Qi Chen, Lin Sun, Zhixin Wang, Kui Jia, and Alan Yuille. “Object as Hotspots: An Anchor-Free 3D Object Detection Approach via Firing of Hotspots”. In: *arXiv: 1912.12791 [cs.CV]* (2019). arXiv: 1912.12791 [cs.CV] (cit. on pp. 124, 126, 127).

[54]Yang Chen and Gérard Medioni. “Object modelling by registration of multiple range images”. In: *Image and vision computing* 10.3 (1992), pp. 145–155 (cit. on pp. 132, 133).

[55]Hao Cheng, Yao Li, and Monika Sester. “Pedestrian group detection in shared space”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 1707–1714 (cit. on p. 41).

[56]François Chollet. “Xception: Deep learning with depthwise separable convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258 (cit. on p. 64).

[58]Phuong Chu, Seoungjae Cho, Sungdae Sim, Kiho Kwak, and Kyungeun Cho. “A Fast Ground Segmentation Method for 3D Point Cloud.” In: *Journal of information processing systems* 13.3 (2017) (cit. on p. 33).

[59]Dorin Comaniciu and Peter Meer. “Mean shift: A robust approach toward feature space analysis”. In: *IEEE Transactions on pattern analysis and machine intelligence* 24.5 (2002), pp. 603–619 (cit. on p. 30).

[60]Eduardo R Corral-Soto et al. “LiDAR few-shot domain adaptation via integrated CycleGAN and 3D object detector with joint learning delay”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 13099–13105 (cit. on pp. 2, 95, 96, 103–107).

[61]Tiago Cortinhal, George Tzelepis, and Eren Erdal Aksoy. “SalsaNext: Fast Semantic Segmentation of LiDAR Point Clouds for Autonomous Driving”. In: *arXiv preprint arXiv:2003.03653* (2020) (cit. on pp. 1, 47, 48, 51, 56, 57, 59, 80, 88, 89, 108).

[62]Henggang Cui. *Technically Speaking: Predicting the future in real time for safer autonomous driving*. Accessed on March 18, 2023. 2022 (cit. on p. 2).

[63]Brian Curless and Marc Levoy. “A volumetric method for building complex models from range images”. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, pp. 303–312 (cit. on p. 97).

[64]Steve DaSilva. *Human Drivers Avoid Crashes 99.999819% of the Time, Self-Driving Cars Need to Be Even Safer*. Jalopnik. Accessed on March 17, 2023. 2023 (cit. on p. 1).

[65]Johannes Deichmann, Eike Ebel, Kersten Heineke, Ruth Heuss, Martin Kellner, and Fabian Steiner. *Autonomous driving’s future: Convenient and connected*. https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected. Accessed on March 17, 2023. McKinsey & Company, 2023 (cit. on p. 1).

[66]Boris Delaunay et al. “Sur la sphere vide”. In: *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk* 7.793-800 (1934), pp. 1–2 (cit. on p. 97).

[67]David DeMers and Garrison Cottrell. “Non-linear dimensionality reduction”. In: *Advances in neural information processing systems* 5 (1992) (cit. on p. 16).

[68]Terrance DeVries and Graham W Taylor. “Improved regularization of convolutional neural networks with cutout”. In: *arXiv preprint arXiv:1708.04552* (2017) (cit. on pp. 80, 82).

[69]Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. “Cswin transformer: A general vision transformer backbone with cross-shaped windows”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12124–12134 (cit. on p. 18).

[70]Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020) (cit. on p. 18).

[71]Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. “CARLA: An open urban driving simulator”. In: *Conference on robot learning*. PMLR. 2017, pp. 1–16 (cit. on p. 131).

[72]Alexey Dosovitskiy et al. “CARLA: An Open Urban Driving Simulator”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16 (cit. on p. 96).

[74]Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. “On the shape of a set of points in the plane”. In: *IEEE Transactions on information theory* 29.4 (1983), pp. 551–559 (cit. on p. 97).

[75]Alberto Elfes et al. “Occupancy grids: A stochastic spatial representation for active robot perception”. In: *Proceedings of the Sixth Conference on Uncertainty in AI*. Vol. 2929. Morgan Kaufmann San Mateo, CA. 1990, p. 6 (cit. on p. 116).

[76]Per K Enge. “The global positioning system: Signals, measurements, and performance”. In: *International Journal of Wireless Information Networks* 1.2 (1994), pp. 83–105 (cit. on p. 119).

[77]Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *Kdd*. Vol. 96. 34. 1996, pp. 226–231 (cit. on pp. 30, 40–43).

[78]Ariane S Etienne, Joëlle Berlie, Joséphine Georgakopoulos, and Roland Maurer. “Role of dead reckoning in navigation.” In: (1998) (cit. on p. 118).

[79]Péter Fankhauser and Marco Hutter. “A universal grid map library: Implementation and use case for rough terrain navigation”. In: *Robot Operating System (ROS)*. Springer, 2016, pp. 99–120 (cit. on p. 116).

[80]Juncong Fei, Kunyu Peng, Philipp Heidenreich, Frank Bieder, and Christoph Stiller. “PillarSegNet: Pillar-based semantic grid map estimation using sparse LiDAR data”. In: *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2021, pp. 838–844 (cit. on pp. 115, 116).

[81]Evelyn Fix and Joseph Lawson Hodges. “Discriminatory analysis. Nonparametric discrimination: Consistency properties”. In: *International Statistical Review/Revue Internationale de Statistique* 57.3 (1989), pp. 238–247 (cit. on p. 99).

[82]Evelyn Fix and Joseph L Hodges Jr. *Discriminatory analysis-nonparametric discrimination: Small sample performance*. Tech. rep. California Univ Berkeley, 1952 (cit. on p. 57).

[83]Tobias Fleck, Lennart Jauernig, Rupert Polley, Philip Schörner, Marc René Zofka, and J Marius Zöllner. “Infra2Go: A Mobile Development Platform for Connected, Cooperative and Autonomous Driving”. In: () (cit. on p. 43).

[84]Keinosuke Fukunaga and Larry Hostetler. “The estimation of the gradient of a density function, with applications in pattern recognition”. In: *IEEE Transactions on information theory* 21.1 (1975), pp. 32–40 (cit. on p. 30).

[85]Stefano Gasperini, Mohammad-Ali Nikouei Mahani, Alvaro Marcos-Ramiro, Nassir Navab, and Federico Tombari. “Panoster: End-to-end panoptic segmentation of lidar point clouds”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3216–3223 (cit. on pp. 62, 73).

[86]Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? the kitti vision benchmark suite”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 3354–3361 (cit. on pp. 35, 40, 126, 138).

[87]Andreas Geiger et al. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2012, pp. 3354–3361 (cit. on pp. 12, 97, 100, 103).

[88]Martin Gerdzhev, Ryan Razani, Ehsan Taghavi, and Bingbing Liu. “TORNADO-Net: mulTiview tOtal vaRiatioN semAntic segmentation with Diamond inceptiOn module”. In: *arXiv preprint arXiv:2008.10544* (2020) (cit. on pp. 47–49).

[89]Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016 (cit. on pp. 14, 16–20).

[90]Heinrich Gotzig and Georg Otto Geduld. “LIDAR-sensorik”. In: *Handbuch Fahrerassistenzsysteme*. Springer, 2015, pp. 317–334 (cit. on pp. 1, 2, 7, 9).

[91]Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. “3d semantic segmentation with submanifold sparse convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 9224–9232 (cit. on pp. 46, 49, 129).

[92]Lukas Hahn. “Aspects of Active Learning, Architecture Search and Lidar Panoptic Segmentation towards Pedestrian Perception in Autonomous Driving”. Dissertation. Bergische Universität Wuppertal, 2022 (cit. on pp. 62, 124).

[93]Lukas Hahn, Frederik Hasecke, and Anton Kummert. “Fast Object Classification and Meaningful Data Representation of Segmented Lidar Instances”. In: *23rd IEEE International Conference on Intelligent Transportation Systems (ITSC)* (2020) (cit. on pp. 62, 64).

[94]Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. “Transformer in transformer”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 15908–15919 (cit. on p. 18).

[95]Stephen Hanson and Lorien Pratt. “Comparing biases for minimal network construction with back-propagation”. In: *Advances in Neural Information Processing Systems* 1 (1988), pp. 177–185 (cit. on pp. 17, 81).

[96]Frederik Hasecke. “Extended Object Detection and Classification with Combined Lidar and Camera Sensor Data”. MA thesis. Bergische Universität Wuppertal, 2020 (cit. on p. 29).

[97]Frederik Hasecke, Martin Alsfasser, and Anton Kummert. “What Can be Seen is What You Get: Structure Aware Point Cloud Augmentation”. In: *2022 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2022, pp. 594–599 (cit. on pp. 80, 137).

[98]Frederik Hasecke, Pascal Colling, and Anton Kummert. “Fake It, Mix It, Segment It: Bridging the Domain Gap Between Lidar Sensors”. In: *Proceedings of the 12th International Conference on Pattern Recognition Applications and Methods*. 2023, pp. 743–750 (cit. on p. 95).

[99]Frederik Hasecke, Lukas Hahn, and Anton Kummert. “FLIC: Fast Lidar Image Clustering”. In: *Proceedings of the 10th International Conference on Pattern Recognition Applications and Methods*. 2021, pp. 25–35 (cit. on pp. 1, 2, 29, 66, 73, 127).

[100]Frederik Lenard Hasecke and Sönke Behrends. *Method, Device, and Computer Program for Determining a Change in Position and/or Orientation of a Mobile Apparatus*. U.S. Patent 20220217499A1, 2021 (cit. on p. 118).

[101]Arthur Hennequin, Lionel Lacassagne, Laurent Cabaret, and Quentin Meunier. “A new Direct Connected Component Labeling and Analysis Algorithms for GPUs”. In: *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. IEEE. 2018, pp. 76–81 (cit. on p. 35).

[102]Franziska Henze, Natalie Magdalena Stasinski, Dennis Fassbender, and Christoph Stiller. “Developers’ Information Needs during Test Drives with Automated Vehicles in Real Traffic: A Focus Group Study”. In: *13th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. 2021, pp. 81–85 (cit. on p. 124).

[103]Hesai. “Pandora All-in-One Sensing Solution for Autonomous Driving - User’s Manual”. In: () (cit. on pp. 8, 12, 34, 90).

[104]Michael Himmelsbach, Felix V Hundelshausen, and H-J Wuensche. “Fast segmentation of 3D point clouds for ground vehicles”. In: *2010 IEEE Intelligent Vehicles Symposium*. IEEE. 2010, pp. 560–565 (cit. on p. 30).

[105]Fangzhou Hong, Hui Zhou, Xinge Zhu, Hongsheng Li, and Ziwei Liu. “Lidar-based panoptic segmentation via dynamic shifting network”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 13090–13099 (cit. on pp. 62, 73).

[106]Andreas Höpe and Kai-Olaf Hauer. “Three-dimensional appearance characterization of diffuse standard reflection materials”. In: *Metrologia* 47.3 (2010), p. 295 (cit. on p. 10).

[107]Joseph Hoshen and Raoul Kopelman. “Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm”. In: *Physical Review B* 14.8 (1976), p. 3438 (cit. on p. 35).

[108]Jordan SK Hu and Steven L Waslander. “Pattern-Aware Data Augmentation for LiDAR 3D Object Detection”. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2021, pp. 2703–2710 (cit. on p. 82).

[109]Juana Valeria Hurtado, Rohit Mohan, Wolfram Burgard, and Abhinav Valada. “Mopt: Multi-object panoptic tracking”. In: *arXiv preprint arXiv:2004.08189* (2020) (cit. on pp. 62, 73).

[110]Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size”. In: *arXiv preprint arXiv:1602.07360* (2016) (cit. on p. 47).

[113]Peng Jiang and Srikanth Saripalli. “LiDARNet: A boundary-aware domain adaptation model for point cloud semantic segmentation”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 2457–2464 (cit. on p. 96).

[114]Philip Joisten, Ziyu Liu, Nina Theobald, Andreas Webler, and Bettina Abendroth. “Communication of automated vehicles and pedestrian groups: An intercultural study on pedestrians’ street crossing decisions”. In: *Proceedings of Mensch und Computer 2021*. 2021, pp. 49–53 (cit. on p. 124).

[115]Rudolph Emil Kalman. “A new approach to linear filtering and prediction problems”. In: (1960) (cit. on pp. 108, 117, 118, 131).

[116]Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. “Poisson surface reconstruction”. In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. Vol. 7. 2006 (cit. on pp. 97, 99, 120, 131–133).

[117]R. Kesten et al. *Level 5 Perception Dataset 2020*. `https://level-5.global/level5/data/`. 2019 (cit. on p. 138).

[118]Motaz Khader and Samir Cherian. “An introduction to automotive lidar”. In: *Texas Instruments* (2020) (cit. on pp. 11, 12).

[119]Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. “Panoptic segmentation”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9404–9413 (cit. on p. 23).

[120]Deyvid Kochanov, Fatemeh Karimi Nejadasl, and Olaf Booij. “KPRNet: Improving projection-based LiDAR semantic segmentation”. In: *arXiv preprint arXiv:2007.12668* (2020) (cit. on pp. 47, 48, 51, 57).

[121]Lingdong Kong, Niamul Quader, and Venice Erin Liong. “ConDA: Unsupervised Domain Adaptation for LiDAR Segmentation via Regularized Domain Concatenation”. In: *arXiv preprint arXiv:2111.15242* (2021) (cit. on pp. 2, 95).

[122]Lingdong Kong, Jiawei Ren, Liang Pan, and Ziwei Liu. “LaserMix for Semi-Supervised LiDAR Semantic Segmentation”. In: *arXiv preprint arXiv:2207.00026* (2022) (cit. on pp. 83, 92, 93).

[123]Dmitriy Korchev, Shinko Cheng, Yuri Owechko, et al. “On real-time lidar data segmentation and classification”. In: *Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV)*. The Steering Committee of The World Congress in Computer Science, Computer . . . 2013, p. 1 (cit. on p. 30).

[124]Hongwu Kuang, Bei Wang, Jianping An, Ming Zhang, and Zehan Zhang. “Voxel-FPN: Multi-scale voxel feature aggregation for 3D object detection from LIDAR point clouds”. In: *Sensors* 20.3 (2020), p. 704 (cit. on pp. 49, 50).

[125]Tin-Yau Kwok and Dit-Yan Yeung. “Constructive algorithms for structure learning in feedforward neural networks for regression problems”. In: *IEEE transactions on neural networks* 8.3 (1997), pp. 630–645 (cit. on p. 16).

[126]Jean Lahoud, Bernard Ghanem, Marc Pollefeys, and Martin R Oswald. “3d instance segmentation via multi-task metric learning”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 9256–9266 (cit. on pp. 30, 31).

[127]Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. “Pointpillars: Fast encoders for object detection from point clouds”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 12697–12705 (cit. on pp. 24, 45–47, 49, 62, 99, 108, 117, 118, 124, 126, 127, 131).

[128]Ferdinand Langer, Andres Milioto, Alexandre Haag, Jens Behley, and Cyrill Stachniss. “Domain transfer for semantic segmentation of LiDAR data using deep neural networks”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 8263–8270 (cit. on pp. 96, 97, 103, 107).

[129]Ltd. Leishen Intelligent System Co. *905nm VS 1550nm: which is better for automotive LiDAR?* Accessed: 2022-12-08. 2022 (cit. on p. 9).

[130]Dan Levi, Noa Garnett, Ethan Fetaya, and Israel Herzlyia. “StixelNet: A Deep Convolutional Network for Obstacle Detection and Road Segmentation.” In: *BMVC*. Vol. 1. 2. 2015, p. 4 (cit. on p. 116).

[131]Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. “Visualizing the loss landscape of neural nets”. In: *Advances in neural information processing systems* 31 (2018) (cit. on p. 16).

[132]Jinke Li, Xiao He, Yang Wen, Yuan Gao, Xiaoqiang Cheng, and Dan Zhang. “Panoptic-PHNet: Towards Real-Time and High-Precision LiDAR Panoptic Segmentation via Clustering Pseudo Heatmap”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 11809–11818 (cit. on pp. 1, 30, 62, 73, 74).

[133]Shijie Li, Xieyuanli Chen, Yun Liu, Dengxin Dai, Cyrill Stachniss, and Juergen Gall. “Multi-scale interaction for real-time lidar data segmentation on an embedded platform”. In: *IEEE Robotics and Automation Letters* 7.2 (2021), pp. 738–745 (cit. on pp. 2, 67, 68, 73, 80, 88, 89).

[134]You Li and Javier Ibanez-Guzman. “Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems”. In: *IEEE Signal Processing Magazine* 37.4 (2020), pp. 50–61 (cit. on p. 12).

[135]Yuanzhi Li and Yingyu Liang. “Learning overparameterized neural networks via stochastic gradient descent on structured data”. In: *Advances in neural information processing systems* 31 (2018) (cit. on p. 17).

[136]Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Qiao Yu, and Jifeng Dai. “BEVFormer: Learning Bird’s-Eye-View Representation from Multi-Camera Images via Spatiotemporal Transformers”. In: *arXiv preprint arXiv:2203.17270* (2022) (cit. on p. 116).

[137]Velodyne Lidar. *A Guide to Lidar Wavelengths for Autonomous Vehicles and Driver Assistance*. Accessed: 2022-12-08. 2021 (cit. on p. 9).

[138]Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755 (cit. on p. 22).

[139]Yuankai Lin, Tao Cheng, Qi Zhong, Wending Zhou, and Hua Yang. “Dynamic spatial propagation network for depth completion”. In: *arXiv preprint arXiv:2202.09769* (2022) (cit. on p. 116).

[140]Charles X Ling and Chenghui Li. “Data mining for direct marketing: Problems and solutions.” In: *Kdd*. Vol. 98. 1998, pp. 73–79 (cit. on p. 81).

[141]Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. “Swin transformer: Hierarchical vision transformer using shifted windows”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 10012–10022 (cit. on p. 18).

[142]Bharat Lohani. “Airborne altimetric LIDAR: Principle, data collection, processing and applications”. In: *URL: http://home. iitk. ac. in/˜ blohani/LiDAR_Tutorial/Airborne_AltimetricLidar_Tutorial. htm, accessed Feb* (2008) (cit. on p. 8).

[143]Jakob Lombacher, Kilian Laudt, Markus Hahn, Jürgen Dickmann, and Christian Wöhler. “Semantic radar grids”. In: *2017 IEEE intelligent vehicles symposium (IV)*. IEEE. 2017, pp. 1170–1175 (cit. on p. 115).

[144]Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440 (cit. on p. 21).

[145]Jiao Mao, Guoliang Xu, Wanlin Li, Xiaohui Fan, and Jiangtao Luo. “Pedestrian detection and recognition using lidar for autonomous driving”. In: *2019 International Conference on Optical Instruments and Technology: Optical Sensors and Applications*. Vol. 11436. International Society for Optics and Photonics. 2020, 114360R (cit. on p. 41).

[146]Margarita Martínez-Díaz and Francesc Soriguera. “Autonomous vehicles: theoretical and practical challenges”. In: *Transportation Research Procedia* 33 (2018). XIII Conference on Transport Engineering, CIT2018, pp. 275–282 (cit. on pp. 1, 2).

[147]Andres Milioto, Jens Behley, Chris McCool, and Cyrill Stachniss. “Lidar panoptic segmentation for autonomous driving”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 8505–8512 (cit. on pp. 62, 73).

[148]Frank Moosmann. *Interlacing self-localization, moving object tracking and mapping for 3d range sensors*. Vol. 24. KIT Scientific Publishing, 2013 (cit. on pp. 39, 40).

[149]Frank Moosmann, Oliver Pink, and Christoph Stiller. “Segmentation of 3D lidar data in non-flat urban environments using a local convexity criterion”. In: *2009 IEEE Intelligent Vehicles Symposium*. IEEE. 2009, pp. 215–220 (cit. on pp. 1, 30, 37).

[152]Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. “Deep double descent: Where bigger models and more data hurt”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2021.12 (2021), p. 124003 (cit. on p. 17).

[153]Lucas Nunes, Xieyuanli Chen, Rodrigo Marcuzzi, Aljosa Osep, Laura Leal-Taixé, Cyrill Stachniss, and Jens Behley. “Unsupervised Class-Agnostic Instance Segmentation of 3D LiDAR Data for Autonomous Vehicles”. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 8713–8720 (cit. on p. 30).

[155]Adam Paszke, Sam Gross, Francisco Massa, et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035 (cit. on p. 71).

[156]F. Pedregosa, G. Varoquaux, A. Gramfort, et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 40).

[157]Kunyu Peng, Juncong Fei, Kailun Yang, et al. “MASS: Multi-attentional semantic segmentation of LiDAR data for dense top-view understanding”. In: *IEEE Transactions on Intelligent Transportation Systems* (2022) (cit. on p. 116).

[158]Lorenzo Porzi, Samuel Rota Bulo, Aleksander Colovic, and Peter Kontschieder. “Seamless scene segmentation”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8277–8286 (cit. on p. 23).

[159]Lutz Prechelt. “Automatic early stopping using cross validation: quantifying the criteria”. In: *Neural networks* 11.4 (1998), pp. 761–767 (cit. on p. 17).

[160]Robert Prophet, Anastasios Deligiannis, Juan-Carlos Fuentes-Michel, Ingo Weber, and Martin Vossiek. “Semantic segmentation on 3D occupancy grids for automotive radar”. In: *IEEE Access* 8 (2020), pp. 197917–197930 (cit. on p. 115).

[161]Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660 (cit. on pp. 1, 24, 46, 50, 68).

[162]Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. “Pointnet++: Deep hierarchical feature learning on point sets in a metric space”. In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 1, 24, 46).

[164]Ryan Razani, Ran Cheng, Enxu Li, Ehsan Taghavi, Yuan Ren, and Liu Bingbing. “GP-S3Net: Graph-based panoptic sparse semantic segmentation network”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 16076–16085 (cit. on pp. 62, 73, 74).

[165]Joseph Redmon and Ali Farhadi. “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018) (cit. on p. 47).

[166]Mrigank Rochan et al. “Unsupervised domain adaptation in lidar semantic segmentation with self-supervision and gated adapters”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 2649–2655 (cit. on pp. 2, 95, 96, 103–107).

[167]R Tyrrell Rockafellar. *Convex analysis*. Vol. 18. Princeton university press, 1970 (cit. on p. 125).

[168]Amir Rosenfeld, Richard Zemel, and John K Tsotsos. “The elephant in the room”. In: *arXiv preprint arXiv:1808.03305* (2018) (cit. on p. 82).

[169]Cristiano Saltori, Fabio Galasso, Giuseppe Fiameni, Nicu Sebe, Elisa Ricci, and Fabio Poiesi. “CoSMix: Compositional Semantic Mix for Domain Adaptation in 3D LiDAR Segmentation”. In: *arXiv preprint arXiv:2207.09778* (2022) (cit. on pp. 96, 102).

[170]Dario D Salvucci. “Modeling driver behavior in a cognitive architecture”. In: *Human factors* 48.2 (2006), pp. 362–380 (cit. on p. 1).

[171]Jules Sanchez, Jean-Emmanuel Deschaud, and François Goulette. “COLA: COarse LAbel pre-training for 3D semantic segmentation of sparse LiDAR datasets”. In: *arXiv preprint arXiv:2202.06884* (2022) (cit. on p. 96).

[172]Maximilian Schäfer, Kun Zhao, Markus Bühren, and Anton Kummert. “Context-aware scene prediction network (caspnet)”. In: *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2022, pp. 3970–3977 (cit. on p. 124).

[173]Marcel Schreiber, Vasileios Belagiannis, Claudius Gläser, and Klaus Dietmayer. “A multi-task recurrent neural network for end-to-end dynamic occupancy grid mapping”. In: *arXiv preprint arXiv:2202.04461* (2022) (cit. on p. 116).

[174]Marcel Schreiber, Vasileios Belagiannis, Claudius Gläser, and Klaus Dietmayer. “Dynamic occupancy grid mapping with recurrent neural networks”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 6717–6724 (cit. on p. 116).

[175]Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. “DBSCAN revisited, revisited: why and how you should (still) use DBSCAN”. In: *ACM Transactions on Database Systems (TODS)* 42.3 (2017), pp. 1–21 (cit. on p. 41).

[176]John Schulman, Barret Zoph, Christina Kim, et al. *ChatGPT: Optimizing Language Models for Dialogue*. 2022 (cit. on p. 13).

[177]Tixiao Shan and Brendan Englot. “LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4758–4765 (cit. on pp. 118, 131).

[178]Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. “From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network”. In: *IEEE transactions on pattern analysis and machine intelligence* 43.8 (2020), pp. 2647–2664 (cit. on pp. 99, 108, 117, 118, 131).

[179]ACM SIGKDD. “2014 SIGKDD Test of Time Award”. In: *https://www.kdd.org/News/view/2014-sigkdd-test-of-time-award* (2014) (cit. on p. 41).

[180]Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on p. 20).

[181]Kshitij Sirohi, Rohit Mohan, Daniel Büscher, Wolfram Burgard, and Abhinav Valada. “Efficientlps: Efficient lidar panoptic segmentation”. In: *IEEE Transactions on Robotics* (2021) (cit. on pp. 62, 73).

[182]Leslie N Smith and Nicholay Topin. “Super-convergence: Very fast training of neural networks using large learning rates”. In: *Artificial intelligence and machine learning for multi-domain operations applications*. Vol. 11006. SPIE. 2019, pp. 369–386 (cit. on pp. 88, 142).

[184]Eduardo D Sontag. “Feedforward nets for interpolation and classification”. In: *Journal of Computer and System Sciences* 45.1 (1992), pp. 20–48 (cit. on p. 16).

[185]Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958 (cit. on pp. 17, 81).

[186]Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, et al. “Scalability in Perception for Autonomous Driving: Waymo Open Dataset”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on p. 138).

[187]Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. “Convolutional neural networks for medical image analysis: Full training or fine tuning?” In: *IEEE transactions on medical imaging* 35.5 (2016), pp. 1299–1312 (cit. on p. 96).

[188]Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. “Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution”. In: *arXiv preprint arXiv:2007.16100* (2020) (cit. on pp. 24, 46, 49).

[189]Andrew Tao, Karan Sapra, and Bryan Catanzaro. “Hierarchical multi-scale attention for semantic segmentation”. In: *arXiv preprint arXiv:2005.10821* (2020) (cit. on p. 96).

[190]OpenPCDet Development Team. *OpenPCDet: An Open-source Toolbox for 3D Object Detection from Point Clouds*. `https://github.com/open-mmlab/OpenPCDet`. 2020 (cit. on p. 136).

[191]Tommaso Toffoli and Norman Margolus. *Cellular automata machines: a new environment for modeling*. MIT press, 1987 (cit. on p. 35).

[192]Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. “Training data-efficient image transformers & distillation through attention”. In: *International conference on machine learning*. PMLR. 2021, pp. 10347–10357 (cit. on p. 18).

[193]Apostolia Tsirikoglou, Joel Kronander, Magnus Wrenninge, and Jonas Unger. “Procedural modeling and physically based rendering for synthetic data generation in automotive applications”. In: *arXiv preprint arXiv:1710.06270* (2017) (cit. on p. 131).

[194]Patrik Vacek, Otakar Jašek, Karel Zimmermann, and Tomáš Svoboda. “Learning to predict lidar intensities”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.4 (2021), pp. 3556–3564 (cit. on pp. 9, 99).

[195]Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 18).

[196]Velodyne. “HDL-64E High Definition Lidar Sensor - User’s Manual”. In: () (cit. on pp. 12, 34).

[197]Velodyne. *HDL32E High Definition LiDAR Sensor - User’s Manual and Programing Guide*. 2015 (cit. on pp. 12, 34).

[198]Velodyne. *VLP-16 LiDAR PUCK - User’s Manual*. 2015 (cit. on p. 34).

[199]Velodyne. *VLS-128 User Manual*. 2018 (cit. on pp. 8, 12, 34).

[201]Pauli Virtanen, Ralf Gommers, Travis E Oliphant, et al. “SciPy 1.0: fundamental algorithms for scientific computing in Python”. In: *Nature methods* 17.3 (2020), pp. 261–272 (cit. on p. 35).

[202]Carina Vogl, Moritz Sackmann, Ludwig Kürzinger, and Ulrich Hofmann. “Frenet coordinate based driving maneuver prediction at roundabouts using LSTM networks”. In: *Proceedings of the 4th ACM Computer Science in Cars Symposium*. 2020, pp. 1–9 (cit. on p. 124).

[204]Ulla Wandinger. “Introduction to lidar”. In: *Lidar*. Springer, 2005, pp. 1–18 (cit. on pp. 2, 8, 9).

[205]Brian H Wang, Wei-Lun Chao, Yan Wang, Bharath Hariharan, Kilian Q Weinberger, and Mark Campbell. “LDLS: 3-D Object Segmentation Through Label Diffusion From 2-D Images”. In: *IEEE Robotics and Automation Letters* 4.3 (2019), pp. 2902–2909 (cit. on p. 31).

[206]Wenhai Wang, Enze Xie, Xiang Li, et al. “Pyramid vision transformer: A versatile backbone for dense prediction without convolutions”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 568–578 (cit. on p. 18).

[207]Yuan Wang, Yang Yu, and Ming Liu. “PointIT: A fast tracking framework based on 3D instance segmentation”. In: *arXiv preprint arXiv:1902.06379* (2019) (cit. on p. 31).

[208]Andreas S Weigend, David E Rumelhart, and Bernardo A Huberman. “Generalization by weight-elimination with application to forecasting”. In: *Advances in Neural Information Processing Systems*. 1991, pp. 875–882 (cit. on pp. 17, 81).

[209]Martin Wermelinger, Péter Fankhauser, Remo Diethelm, Philipp Krüsi, Roland Siegwart, and Marco Hutter. “Navigation planning for legged robots in challenging terrain”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 1184–1189 (cit. on p. 116).

[210]Julian Wiederer, Arij Bouazizi, Ulrich Kressel, and Vasileios Belagiannis. “Traffic control gesture recognition for autonomous vehicles”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 10676–10683 (cit. on p. 124).

[211]Bichen Wu et al. “Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 1887–1893 (cit. on pp. 24, 47).

[212]Aoran Xiao, Jiaxing Huang, Dayan Guan, Kaiwen Cui, Shijian Lu, and Ling Shao. “PolarMix: A General Data Augmentation Technique for LiDAR Point Clouds”. In: *arXiv preprint arXiv:2208.00223* (2022) (cit. on pp. 83, 92, 93).

[213]Aoran Xiao et al. “Transfer learning from synthetic to real LiDAR point cloud for semantic segmentation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 3. 2022, pp. 2795–2803 (cit. on p. 96).

[214]Pengchuan Xiao et al. “Pandaset: Advanced sensor suite dataset for autonomous driving”. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. 2021, pp. 3095–3101 (cit. on p. 138).

[215]Chenfeng Xu, Bichen Wu, Zining Wang, Wei Zhan, Peter Vajda, Kurt Keutzer, and Masayoshi Tomizuka. “Squeezesegv3: Spatially-adaptive convolution for efficient point-cloud segmentation”. In: *arXiv preprint arXiv:2004.01803* (2020) (cit. on p. 48).

[216]Yan Yan, Yuxing Mao, and Bo Li. “Second: Sparsely embedded convolutional detection”. In: *Sensors* 18.10 (2018), p. 3337 (cit. on pp. 81, 82, 124, 126, 127, 138).

[217]Bo Yang, Jianan Wang, Ronald Clark, Qingyong Hu, Sen Wang, Andrew Markham, and Niki Trigoni. “Learning object bounding boxes for 3d instance segmentation on point clouds”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 6740–6749 (cit. on pp. 30, 31).

[218]De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. “Sensor and sensor fusion technology in autonomous vehicles: A review”. In: *Sensors* 21.6 (2021), p. 2140 (cit. on pp. 11, 12).

[219]Li Yi, Boqing Gong, and Thomas Funkhouser. “Complete & label: A domain adaptation approach to semantic segmentation of lidar point clouds”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 15363–15373 (cit. on p. 96).

[220]Dimitris Zermas, Izzat Izzat, and Nikolaos Papanikolopoulos. “Fast segmentation of 3d point clouds: A paradigm on lidar data for autonomous vehicle applications”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 5067–5073 (cit. on pp. 1, 2, 30).

[221]Chi Zhang, Guosheng Lin, Fayao Liu, Rui Yao, and Chunhua Shen. “Canet: Class-agnostic segmentation networks with iterative refinement and attentive few-shot learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5217–5226 (cit. on p. 22).

[222]Feihu Zhang, Jin Fang, Benjamin Wah, and Philip Torr. “Deep fusionnet for point cloud semantic segmentation”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIV 16*. Springer. 2020, pp. 644–663 (cit. on p. 46).

[223]Feihu Zhang, Chenye Guan, Jin Fang, Song Bai, Ruigang Yang, Philip Torr, and Victor Prisacariu. “Instance segmentation of lidar point clouds”. In: *ICRA* 4.1 (2020) (cit. on pp. 30, 31).

[224]Shuyang Zhang, Fulong Ma Di Wang, Chao Qin, Zhengyong Chen, and Ming Liu. “Robust Pedestrian Tracking in Crowd Scenarios Using an Adaptive GMM-based Framework”. In: () (cit. on p. 41).

[225]Yang Zhang, Zixiang Zhou, Philip David, Xiangyu Yue, Zerong Xi, Boqing Gong, and Hassan Foroosh. “Polarnet: An improved grid representation for online lidar point clouds semantic segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9601–9610 (cit. on p. 68).

[226]Sicheng Zhao et al. “ePointDA: An end-to-end simulation-to-real domain adaptation framework for LiDAR point cloud segmentation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 4. 2021, pp. 3500–3509 (cit. on p. 96).

[227]Yiming Zhao, Xiao Zhang, and Xinming Huang. “A divide-and-merge point cloud clustering algorithm for LiDAR panoptic segmentation”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 7029–7035 (cit. on p. 30).

[228]Hui Zhou, Xinge Zhu, Xiao Song, Yuexin Ma, Zhe Wang, Hongsheng Li, and Dahua Lin. “Cylinder3D: An Effective 3D Framework for Driving-scene LiDAR Semantic Segmentation”. In: *arXiv preprint arXiv:2008.01550* (2020) (cit. on pp. 1, 24, 46, 49, 67, 68, 73, 80, 88–93, 104–107, 117, 131).

[229]Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. “Open3D: A Modern Library for 3D Data Processing”. In: *arXiv:1801.09847* (2018) (cit. on p. 99).

[230]Yin Zhou and Oncel Tuzel. “VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4490–4499 (cit. on pp. 24, 46, 124, 126, 127).

[231]Zhi-Hua Zhou. “Why over-parameterization of deep neural networks does not overfit?” In: *Science China Information Sciences* 64 (2021), pp. 1–3 (cit. on p. 17).

[232]Zixiang Zhou, Yang Zhang, and Hassan Foroosh. “Panoptic-polarnet: Proposal-free lidar point cloud panoptic segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 13194–13203 (cit. on pp. 24, 62, 73, 82).

Webpages

[@33]Aptiv. *What Are the Levels of Automated Driving?* Accessed: 2023-02-12. 2020. URL: https://www.aptiv.com/en/insights/article/what-are-the-levels-of-automated-driving (cit. on p. 130).

[@43]*Bicycle Free 3D model*. kyumamoon. URL: https://www.cgtrader.com/free-3d-models/vehicle/bicycle/bicycle-36e28aba-d615-4c71-b4d6-c74782b673a3 (cit. on pp. 138, 139).

[@57]Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras (cit. on p. 66).

[@73]*Dutch Bike Free 3D model*. eslamovich. URL: https://www.cgtrader.com/free-3d-models/vehicle/bicycle/dutch-bike (cit. on pp. 138, 139).

[@111]Innoviz. *InnovizTwo Website*. Accessed: 2022-09-05. 2022. URL: https://innoviz.tech/innoviztwo (cit. on pp. 12, 136, 138).

[@112]Innoviz and NVIDIA. *ECCV workshop on 3D Perception for Autonomous Driving: The LiDAR Self-Supervised Learning Challenge: Learning From a Limited Amount of High-Resolution LiDAR data.* Accessed: 2022-09-15. 2022 (cit. on pp. 12, 136, 137, 146).

[@150]*Moped Free low-poly 3D model*. Goran Dodic. URL: https://www.cgtrader.com/free-3d-models/vehicle/motorcycle/moped (cit. on pp. 138, 139).

[@151]*motorcycle Free 3D model*. qq577148845. URL: https://www.cgtrader.com/free-3d-models/vehicle/bicycle/motorcycle-fcfd399a-5552-4d55-806b-9eb459f3887e (cit. on pp. 138, 139).

[@154]*NVIDIA Showcases Novel AI Tools in DRIVE Sim to Advance Autonomous Vehicle Development*. NVIDIA. URL: https://blogs.nvidia.com/blog/2022/03/23/drive-sim-omniverse-neural-ai-digital-twin/ (cit. on p. 134).

[@163]*Rain Rig*. Blender Foundation. URL: https://studio.blender.org/characters/5f1ed640e9115ed35ea4b3fb/v2/ (cit. on pp. 138, 139).

[@183]*Snow Rig*. Blender Foundation. URL: https://studio.blender.org/characters/snow/v2/ (cit. on pp. 138, 139).

[@200]*Vincent Rig*. Blender Foundation. URL: https://studio.blender.org/characters/5718a967c379cf04929a4247/v1/ (cit. on pp. 138, 139).

[@203]*Wabi Lightning SE Fixed Gear Bicycle Free 3D model*. Semyon Filippov. URL: https://www.cgtrader.com/free-3d-models/vehicle/bicycle/wabi-lightning-se-fixed-gear-bicycle (cit. on pp. 138, 139).

Acronyms

ADAS Advanced Driver Assistance Systems.

CNN Convolutional Neural Network.

CPU Central Processing Unit.

DBSCAN Density Based Spatial Clustering of Applications with Noise.

FLIC Fast Lidar Image Clustering.

FN False Negative.

FP False Positive.

GPU Graphics Processing Unit.

GT Ground Truth.

Laser Light Amplification by Stimulated Emission of Radiation.

LED Light Emitting Diode.

Lidar Light Detection and Ranging.

MSE Mean Squared Error.

Radar Radio Detection and Ranging.

RGB Red, Green and Blue.

.

SAPCA Structure Aware Point Cloud Augmentation.

SGD Stochastic Gradient Descent.

TP True Positive.

VFE Voxel Feature Encoding.

Glossary

Ablation Study Analysis of a model's performance after removing components/features.

Augmentation Technique to generate new data from existing data.

Average IoU Average Intersection over Union over all instance (see Page 22).

Average Precision A metric used to evaluate the accuracy of object detection (see Page 142).

Closed Loop A control system where the output affects the input.

Clustering Grouping similar data points into clusters.

Domain Adaptation Adapting a model trained on one domain to another domain.

Ego Vehicle The vehicle the sensor is mounted on.

Instance Segmentation Segmenting individual objects.

IoU Intersection over Union (see Page 21).

Map Connection Connection between pixels beyond the direct pixel neighborhood.

mean Average Precision The average AP value over multiple object classes (see Page 142).

Mesh A three-dimensional model consisting of vertices, edges, and faces.

mIoU Mean Intersection over Union over all classes (see Page 22).

Non-causal Not limited to a particular time sequence or order.

NuScenes A large-scale autonomous driving dataset with various sensor modalities.

Online Making predictions on new data in real-time.

Panoptic Segmentation Combining semantic and instance segmentation.

Pillar A voxel in a 2D grid.

Point Cloud Collection of data points in three-dimensional space.

PQ Panoptic Quality, combining instance and semantic information (see Page 23).

Pseudo Labels Labels generated by a model for unlabeled data.

RangePillar Voxels in a spherical coordinate system.

Real-time Processing data and producing output with minimal delay.

Recall The proportion of true positive samples correctly identified (see Page 42).

Scout Connection Connection between pixels very far beyond the direct pixel neighborhood.

Segmentation Dividing an object into distinct parts or regions.

Self-supervised Training a model with unlabeled data via pretext tasks.

Semantic Segmentation Segmenting data into meaningful parts (e.g., objects).

SemanticKITTI A large-scale outdoor semantic segmentation dataset.

Semi-supervised Training a model with both labeled and unlabeled data.

State of the Art Current highest level of performance on a task.

Supervised Training a model with labeled input-output pairs.

Training Teaching a model to make predictions from input data.

Unsupervised Training a model with unlabeled data without any supervision.

Voxel A three-dimensional pixel or volumetric pixel.

List of Figures

List of Tables